SMART CARD FAULT ATTACKS

ON PUBLIC KEY AND ELLIPTIC CURVE CRYPTOGRAPHY

A Thesis

Submitted to the Faculty

of

Purdue University

by

Jie Ling

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Electrical and Computer Engineering

May 2014

Purdue University

Indianapolis, Indiana

ACKNOWLEDGMENTS

This is the acknowledgment of my Master thesis. The people I appreciate will not be limited in the list below.

It is my great privilege to be here, ECE Department, Indiana University-Purdue University, Indianapolis. At the closing of my Master's program, I find there is not any other moment in my 30 years life being so precious and challenging. This is my first engineering degree. During the past two years, I owe too much from those who provided me strength and confidence.

My first debt of gratitude must go to Dr. Brian King, my thesis advisor, for all I have learned from him and for his continuous help in all stages of this thesis. I want to thank him for his patience and support not only in my study, research work, and also future career. Those skills and experiences gained in researching with him will be a big plus in for my future work. I am also appreciated him for the freedom that he provided to pursue independent work, for being a open person to ideas, for encouraging and helping shaping my interest and ideas. From him, on top of his strong background in this area, the prudent attitude towards research inspired me all through my graduate study and work.

Special thanks to my committee, Dr. Sankook Lee and Dr. Paul Salama. Thanks for their guidance and support. Dr. Sankook Lee, thank you for carefully pointing out all those errors I made in the writing. Your suggestion that to compare the efficiency brought a new perspective to our new algorithms. Thank you, Dr. Salama, for your suggestions on my presentation and the errors I made there, which is so valuable and help me corrected my misunderstanding on an important concept.

Our department coordinator Sherrie Tucker, who is always supportive and patient, from whom I got all kinds of help, study, work, as well as this thesis. Sherrie, thank you for your careful reading on this paper. The corrections you suggested tells me how a more standard thesis should be.

In the end, I want to express my thanks to Dr. Brian King, Dr. Steven Rovnyak, Mr. Amin Hagyousif, Dr. Ji Ronghui, Dr. John Lee, Dr. Lauren Christopher, Dr. Paul Salama, Dr. Sankook Lee and Mr. Daniel Longbottom, for your classes and knowledge I leaned from you. Thank you my parents for your continues love and supporting my decisions. Thank you Shreya and Saleh Almaster for your generous help in both study and living.

Thank you IUPUI. You are the best and warmest school. Thank you Indiana. You possess the most friendly people here. Thank you everyone who contribute to this beautiful place where we study.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Ling, Jie M.S.E.C.E, Purdue University, May 2014. Smart Card Fault Attacks on Public Key and Elliptic Curve Cryptography. Major Professor: Brian King.

Blömmer, Otto, and Seifert presented a fault attack on elliptic curve scalar multiplication called the Sign Change Attack, which causes a fault that changes the sign of the accumulation point. As the use of a sign bit for an extended integer is highly unlikely, this appears to be a highly selective manipulation of the key stream. In this thesis we describe two plausible fault attacks on a smart card implementation of elliptic curve cryptography.

King and Wang designed a new attack called counter fault attack by attacking the scalar multiple of discrete-log cryptosystem. They then successfully generalize this approach to a family of attacks. By implementing King and Wang's scheme on RSA, we successfully attacked RSA keys for a variety of sizes. Further, we generalized the attack model to an attack on any implementation that uses NAF and wNAF key.

1. INTRODUCTION

Sandy, an account processor receives a phone call from a women, saying that someone has being attempted to steal his identity. He inquires to an identity protection service and asks for his name, date of birth and social security number from Sandy, who is still not aware that his identity is going to be stolen. A week later, Sandy's credit card gets declined at the gas station for the reason of an overdraft on his account. Someone from Florida is enjoying new jewelry, jet ski and luxury cosmetics from the replica of Sandy's credit card. This is the opening scene of the Hollywood movie "Identity Thief", which made 174 million dollars box office. The story moves on as a comedy that the mid-mannered man travels across the country to confront this harmless looking women who lives on credit card scams. Although the theme of this film is to tell people life tips, there is a genuine tension about credit card security throughout.

This film, reminds everyone of the risks for which our convenient consuming approach has brought to us. Since smart cards came in to our everyday life, they have become a popular security token, implemented in a number of applications with diverse security requirements, e.g, credit cards, bank cards, cellular communications, electronic cash, banking, satellite TV and Government identifications.

The first smart card fraud, reported in 1995, was the villain tried to register fuel sales on cloned cards to get rebate. This caused thirty million dollars loss to the card issuer [21]. In 1999, software engineer Serge Humpinch, announced his attack on public key system of French Cartes Bancaire. He claimed the success by purchasing 10 Metro tickets from an online system, then asked the bank for 28 million dollars for his scheme. This man was arrested immediately for blackmail [7].

Though such examples appear dated, the threat today is very real. Furthermore, recent advances in attack scenarios bring greater threats. In 1996, Kocher from MIT

brought *timing attack* to the world, which pioneered the later generated power monitoring attack, electromagnetic attack, etc. His work lays the foundation of many powerful side channel attacks. His techniques can efficiently break public key cryptosystems. Thus a new industry was created to protect smart cards, the reason is that smart cards are embedded in a number of highly sensitive and high-financial impacted applications.

Smart cards used to be touted as "secure"' portable devices. In high end applications, like banking and identification applications, the smart card may be used to provide a signature and involves a secret signing key. The key placed on the smart card is usually owned by the card issuer rather than the card possessor, e.g. even if a hacker processed a million credit cards and technically "dissected" them, the key would still lay in the bank systems "safely".

Nevertheless, curiosity about the security of smart cards has not stopped. These cards could emit two contradictory signals to people: they can be attacked and they are secure. While the answer is a smart card system has proven to be more reliable than other machine based cards, such as magnetic stripe and barcode. However this reliance does not withstand all attempts to break the secret, as smart card is not impeccable. Various adversarial attacks have been developed in attempts to determine the secret key. Once the signing key is known to the adversary they can construct a signature and bypass the necessary procedure, a result in a banking application could imply unlimited access to the banks financial resources. But how, you will ask. Actually it is everywhere, and probably you are already targeted.

In this thesis we discuss several smart card "fault attack". In Chapter 2, we provide some background information. In Chapter 3 we discuss smart card security and introduce several fault attacks. In Chapters 4 and 5 we discuss our fault attacks.

2. MATHEMATICS/CRYPTOGRAPHIC BACKGROUND

A *field* is a nonempty set $\mathbb{F}$ with two operations: addition and multiplication such that $\mathbb{F}$ is a commutative additive group and all nonzero elements of $\mathbb{F}$ form a commutative multiplicative group. Further the multiplication distributes over the addition. In a computational setting, one needs a discrete set, so one typically uses a *finite field*.

An *encryption algorithm* is an operation that is used to ensure privacy. The message to be encrypted is called plaintext. The encrypted message is called the ciphertext. Encryption will utilize some "encryption key" k_e. The encryption must be reversible. Decryption is the operation that reverses encryption. Decryption will use a decryption key k_d which must be a secret key. We collectively call the encryption and decryption schemes together with the encryption and decryption keys, a *cryptosystem*.

A cryptosystem for which it is easy to compute the decryption key given the encryption key is called a *symmetric cryptosystem*. A cryptosystem for which it is hard to compute the decryption key given the encryption key is called an *asymmetric-key cryptosystem* or *public-key cryptosystem*. The encryption key is the public-key and will be denoted by pk, whereas the decryption key must be secret and will be denoted by sk. Public-key cryptosystems are often based on some type of hard problem like *factoring* or *discrete log problem*. A design of a cryptosystem is useful as long as the method to break it is infeasible. An easily broken system is not reliable. Many schemes are based on problems that are computationally "hard", namely computational secure problem. For example the problem of factoring large integers is considered to be computationally "hard". This thesis concerns public-key cryptography in the context of its use on smart cards.

There are a number of functions that are used to increase security, such as MAC (message authentication codes), signatures and hash functions. A hash function h is an algorithm to maps data of arbitrary length to data of fixed length w. Moreover, it needs to possess the following property:

1. $h(m)$ is "efficient to compute".
2. On any input x the output $h(x)$ should be computationally indistinguishable from a uniform binary string in the interval $[0, 2^w)$.
3. Preimage resistance - for essentially all pre-specified outputs, it is computationally infeasible to find any input which hashes to that output.
4. 2nd-preimage resistance - it is computationally infeasible to find any second input which has the same output as any specified input.
5. Collision resistance - it is computationally infeasible to find any two distinct inputs x, x' which hash to the same output.

Hash functions are used in a number of security applications. Common hash functions include SHA1 [29](which is obsolete), SHA2 [28], RIPEMED, and several in the MD family [20].

The Secure Hash Algorithm, known as the U.S. Federal Information Processing Standard (FIPS), is a group of hash functions designed published by the the U.S. National Security Agency (NSA) and then published by NIST(National Institute of Standards and Technology). SHA-1 [29] was designed as a hash function with 160-bit argument, a weakness was found in SHA1 and it is no longer in wide use today. SHA-2 [28] is believed to be much stronger and includes a significant number of changes from its predecessor, SHA-1. SHA-2 currently consists of six hash functions with digests that are 224, 256, 384 or 512 bits.

A digital signature is the cryptographic application that models real-life signature. There are two phases the signature phase and the verify phase (see Fig 2.1 and Fig 2.2).

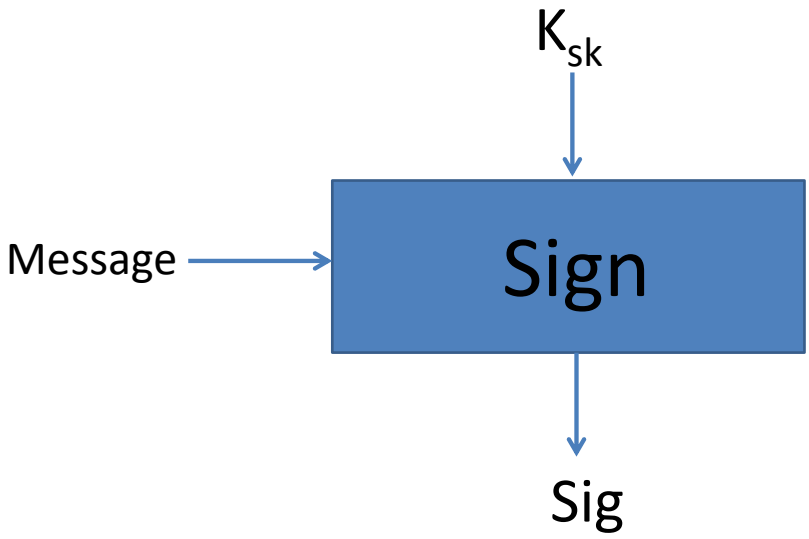


Fig. 2.1.: Signing Phase

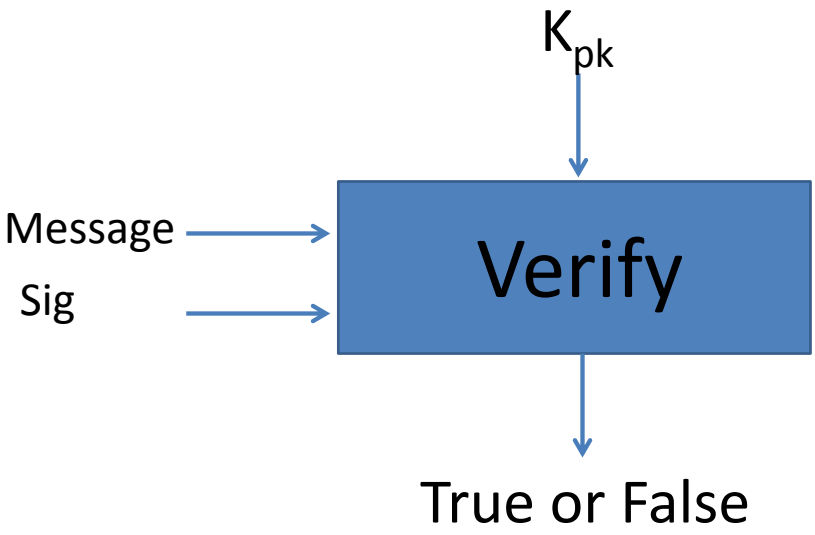


Fig. 2.2.: Verification Phase

A digital signature consists of the following $(M, S, \mathcal{K}, \mathcal{K}', \mathsf{KEY_{GEN}}, \mathsf{SIGN}, \mathsf{VERIFY})$ where:

- M is the plaintext message space.
- S is the signature space.

- $\mathcal{K}$ (set of possible keys for signature generation) is the signing key space.
- $\mathcal{K}'$ (a set of possible verification keys) is the verification key space.
- $KEY_{Gen} : N \to \mathcal{K} \times \mathcal{K}'$ is an efficient key generation algorithm.
- $SIGN : M \times \mathcal{K} \to S$ is an efficient signing algorithm.
- $VERIFY : S \times M \times \mathcal{K}' \to \{\text{true}, \text{false}\}$ is an efficient verification algorithm.

A digital signature Sig is a proof that the signer has authenticated the signed message. A concern is whether other parties can create forged signatures. Thus the signing key k_{sig} is the secret key sk. Anyone can verify the signature, so the verify key k_{ver} is the public key pk. Common signatures schemes are RSA [27], DSS [22], El Gamal [10] and ECDSA [1]. The RSA signature scheme is the scheme that is most commonly used today.

2.1 RSA

In [8], Diffie and Hellman publicly suggested a need for a public-key encryption. However they did not provide a public key-encryption method. In [27] Rivest, Shamir, and Aldeman proposed a public key encryption method based on the difficulty of factoring. Today, this is the most common public-key encryption used (and most common digital signature algorithm used).

In the RSA cryptosystem, there are two keys, one is the public key (e, N), used for encryption and publicly available, the other is the private (secret) key d used for decryption. The cryptosystem is determined in the following manner. Suppose Bob would like to set up a RSA cryptosystem. Bob chooses two distinct prime numbers p and q that are suitably large. Bob set $N = p \cdot q$. the Euler totient function $\varphi(N) = (p-1) \cdot (q-1)$. Bob selects his public key e so that:

$$gcd(e, \varphi(N)) = 1.$$

Thus e has a multiplicative inverse in $\mathbb{Z}_{\varphi(N)}$. Then Bob's secret key d is computed as:

$$d = e^{-1} \mod \varphi(N).$$

Bob's public key is officially the pair (e, N).

Now if Alice would like to send a message m to Bob privately, she locates Bob's public key (e, N) and computes the ciphertext C by $C = m^e \mod N$ (see Algorithm 1) and sends C to Bob.

Bob decrypts C by computing:

$$m = C^d \mod N.$$

The security of RSA is related to keeping d secret. Since e is public, it is necessary to keep $\varphi(N)$ secret (thus it is essential to keep both p and q secret). Further N is made public and $N = p \cdot q$. So N must be large enough that factoring algorithms cannot be applied. In addition, since the factorization of N is dependent on its smallest factors, the primes p and q are selected to be roughly the same bit size [14] (see Fig 2.3 for reference concerning the size of keys).

Suppose the binary representation of e is given by $e = e_{w-1}e_{w-2}\dots e_1e_0$ then $m^e \mod N$ can be calculated as:

Algorithm 1 Compute $m^e \mod N$

1: Set $C \leftarrow 0, i \leftarrow w - 1$
2: **while** $i \geq 0$ **do**
3: $\quad C \leftarrow C^2 \mod N$
4: $\quad$ **if** $e_i \leftarrow 1$ **then**
5: $\quad\quad C \leftarrow C \cdot m \mod N$
6: $\quad$ **end if**
7: $\quad i \leftarrow i - 1$
8: **end while**
9: **Output** C

Date	Minimum of Strength	Symmetric Algorithms	Asymmetric	Discrete Logarithm		Elliptic Curve	Hash (A)	Hash (B)
				Key	Group			
2010 (Legacy)	80	2TDEA*	1024	160	1024	160	SHA-1**	SHA-1
							SHA-224	SHA-224
							SHA-256	SHA-256
							SHA-384	SHA-384
							SHA-512	SHA-512
2011 - 2030	112	3TDEA	2048	224	2048	224	SHA-224	SHA-1
							SHA-256	SHA-224
							SHA-384	SHA-256
							SHA-512	SHA-384
								SHA-512
> 2030	128	AES-128	3072	256	3072	256	SHA-256	SHA-1
							SHA-384	SHA-224
							SHA-512	SHA-256
								SHA-384
								SHA-512
>> 2030	192	AES-192	7680	384	7680	384	SHA-384	SHA-224
							SHA-512	SHA-256
								SHA-384
								SHA-512
>>> 2030	256	AES-256	15360	512	15360	512	SHA-512	SHA-256
								SHA-384
								SHA-512

Fig. 2.3.: Cryptographic Key Length Recommendation

2.1.1 RSA Signing

To sign a message M using RSA, you first compute the hash of M, $m = h(M)$. You sign the message M by computing Sig as:

$$Sig = m^d \bmod N,$$

where d is the secret key. The message M and Sig are given to the entity(s) that need the signature together with the signers public key (e, N). They can verify the signature by:

$$\text{comparing } h(M) \text{ with } Sig^e \mod N.$$

If they are the same the signature is good, otherwise it is bad.

2.2 Elliptic Curve Cryptography (ECC)

An elliptic curve, in the context of public-key cryptography, is a finite Abelian additive group with a large prime subgroup [4, 11]. It is defined over a finite field, typically over a prime field $\mathbb{Z}_p$ or the Galois field (binary field) $GF(2^m)$. In general, let $\mathbb{F}$ be a finite field, an elliptic curve E consists of all points $(x, y) \in \mathbb{F} \times \mathbb{F}$, together with a special point $\mathcal{O}$ (called the point of infinity), that satisfy:

$$y^2 + a_1xy = x^3 + a_2x^2 + a_3x + a_4.$$

Here there is a natural addition that can be executed where $\mathcal{O}$ is the additive identity. For all points $P \in E$ there exists a point $-P \in E$ such that $P + -P = \mathcal{O}$. Note the computation of a negative point $-P$, depends on the field used. Suppose $P = (x, y)$. If the field that defines E is $\mathbb{Z}_p$, then $-P = (x, -y)$. If the field that defines E is $GF(2^m)$, then $-P = (x, x + y)$.

The fundamental cryptographic computation in an elliptic curve cryptography (ECC) is the scalar multiple of a point. That is, given a fixed public point P, that belongs to the prime subgroup, compute the *scalar multiple* kP (which is $P + P + \cdots + P = kP$). Let q denote the prime order of this large subgroup and let $k = k_nk_{n-1} \cdots k_2k_1k_0$ denote the binary representation of k, then the algorithm to compute kP is provided in Algorithm 2.

The *Elliptic Curve Discrete Log Problem* is such that given $(\mathbb{F}, E, q, P, kP)$ it is computationally hard to determine k. For security purposes the subgroup size should be suitably large. In practice, a 160+ bit group size would provide a security level equivalent to 1000+ bit RSA. For a highly sensitive application, like banking, one would require larger keys, such as 220+ bit ECC key, equivalent to a 2000+ bit RSA key. In practice, it is not wise for one to generate their own elliptic curve, but fortunately there are precomputed elliptic curve that have been prescribed by the federal government [23].

ECC is often the cryptographic choice when one is working with in a constrained computing environment, especially where signature generation is required. The essential calculation in ECC is the scalar multiple as described in Algorithm 2. It is important for the smart card to safeguard the key k from side channel attacks.

Here we describe an active side channel attack, a fault attack, where the adversary will enact many fault attacks, collect the outputs and off-line use these faulty outputs to determine the key. The adversary does not know where in the execution process the attack occurs, just the type of attack.

Algorithm 2 Scalar Multiple

input E, P, k

output kP

1: $Q = \mathcal{O}$
2: $i = n$
3: **while** $i \geq 0$ **do**
4: $\quad Q = 2Q$
5: $\quad$ **if** $k_i = 1$ **then**
6: $\quad\quad Q = Q + P$
7: $\quad$ **end if**
8: $\quad i = i - 1$
9: **end while**
10: **output** Q

2.2.1 ECDSA

The Elliptic Curve Digital Signature Algorithm (ECDSA) offers a variant of the Digital Signature Algorithm (DSA) with Elliptic Curve Cryptography, which is widely used.

Key Size

A security level of 112 bits implies the attacker needs the equivalent of about 2^{112} operations to recover the private key, the size of a RSA public key is 2048 bits, whereas ECDSA public key requires only 224 bits. On the other hand, the size of the signature is identical for RSA and ECDSA, which is 4 times of the security level, around 320 bits (see Table [14]).

Signing Algorithm

Alice is to send a message to Bob. Both of them must agree on the parameters of this algorithm, the Curve, P, and n.

- **Curve** The Elliptic Curve, which is in fact the field, to be involved.
- **P** The base point P on the ECC, also known as the generator, with large order.
- **n** The order of P, usually a prime, such that $nP = \mathcal{O}$.

The digital signature follows the following steps: [1]

1. Do $e = \text{HASH}(m)$, where HASH is a cryptographic hash function from the SHA family, eg, SHA-2.
2. Let z be the L_n leftmost bits of e, where L_n is the bit length of the group order n.
3. Select a random integer k from $[1, n-1]$.
4. Calculate the curve point $(x_1, y_1) = k \times P$.
5. Calculate $r = x_1 \mod n$. If r = 0, go back to step 3.
6. Calculate $s = k^{-1}(z + rd_A) \mod n$. If $s = 0$, go back to step 3. The signature is the pair (r, s).

Note the signature requires a scalar multiple calculation.

Verification Algorithm

Bob is to verify Alice's signature, he must have Alice's public key, a curve point Q_A, which is need to be authenticated too: [1]

1. Check that Q_A is not equal to the identity element $\mathcal{O}$, and its coordinates are otherwise valid.
2. Check that Q_A lies on the curve.
3. Check that $nQ_A = \mathcal{O}$.

Then Bob does the following:

1. Verify that r and s are integers in $[1, n-1]$. If not, the signature is invalid.
2. Calculate $e = \text{HASH}(m)$, where HASH is the same function used in the signature generation.
3. Let z be the L_n leftmost bits of e.
4. Calculate $w = s^{-1} \mod n$.
5. Calculate $u_1 = zw \bmod \text{n}$ and $u_2 = rw \bmod \text{n}$.
6. Calculate the curve point $(x_1, y_1) = u_1 \times P + u_2 \times Q_A$.
7. The signature is valid if $\text{r} \equiv x_1 \pmod n$, invalid otherwise.

2.3 Computational Complexity of Public Key Algorithms

When implementing cryptography in a constrained environment, it is common to explore the use of implementing speedups. The security of a Public Key Cryptographic algorithm mainly depends on the computational complexity of their hardest components. For example, the hardness of Diffie Hellman relies on the speed of discrete logarithm. Most of these public key systems involve computational hard operations, which are much more expensive than symmetric cryptosystems, such as block ciphers.

Public-key cryptography can be computationally intensive, so often is constrained environments, one attempts to reduce intensive computations. To speed up the system, the intuitive way is to accelerate the base computation. For example, when using RSA, one may decide to use the Chinese Remainder Theorem (CRT) [31]. On average, such an implementation will reduce the running time to 1/4 of a RSA calculation, because it reduces a $n-$bits exponentiation to two $\frac{n}{2}$ bit exponentiations.

Further in RSA one can explore reducing the computational complexity of the exponentiation of $m^e \mod N$. According to Algorithm 1, the bit length is the number of iterations of the loop in this algorithm. Hence a short bit-length and small Hamming weight result in more efficient encryption. Smaller values of e (such as 3) have been shown to be insecure in some settings.

Choosing a small value makes encryption fast to compute but if it is too small where cipher text is lower than N. Since e is the public exponent it does not need to be selected randomly. On the other hand, the value of d must be secret, so it must be large and random. However, the most common use of RSA, especially in a constrained environment, is as a signature scheme and the signing key is d.

An alternate idea is to reduce the hamming weight of key, expressing the key in *non-adjacent form* (NAF) is a common way [26]. In this form, an integer is expressed as $a_{w-1} \ldots a_1 a_0$ such that each symbol a_i belongs to $\{-1, 0, 1\}$ and no two consecutive symbols $a_{i=1}$ and a_i are nonzero. The use of a signed digit representation of the cryptographic key can reduce the number of complex computations that are needed to perform the cryptographic operation, which results in reduced power consumption. The non-adjacent form (NAF) of a number is a unique signed digit representation. For example, the binary representation of the integer 622 is 1,0,0,1,1,0,1,1,1,0, and it can be computed as $1, 0, 1, 0, 0, -1, 0, 0, -1, 0,$ which is $512 + 128 - 16 - 8$. By expressing a key in NAF form, one will reduce the number of nonzero symbols, it has been shown that the expected percentage of nonzero symbols of a key in NAF form is 1/3 [3].

Algorithm 3 Reitwiesner's canonical recoding [13, 26]

1: **INPUT:** $d_n, ..., d_0$

2: **OUTPUT:** $\delta_{n+1}, ..., \delta_0$

3: $c_0 \leftarrow 0$

4: **for** $j = 0$ to $n + 1$ **do**

5: $\quad c_{j+1} \leftarrow \lfloor (d_j + d_{j+1} + c_j)/2 \rfloor$

6: $\quad \delta_j \leftarrow d_j + c_j - 2c_{j+1}$

7: **end for**

8: **RETURN** $\delta_{n+1}\delta_{n-1} \dots \delta_0$

In many algebraic settings the inverse is very efficient to compute. This is especially true when using elliptic curve cryptosystems (ECC), where the inverse (negative) of an ECC point can be trivially computed from the ECC point. Thus the the use of the symbol -1 in the NAF form is computationally efficient.

2.3.1 wNAF

By using non-adjacent form NAF form, the number of non zeros is to be reduced by 1/6. One can make even further saving on calculation by using wNAF form. "w" here stands for "window". We can also interpret wNAF as, there is not adjacent non zero bits in any consecutive w symbols. Formally a key in wNAF form is:

Definition 1. *A sequence of signed bits is called wNAF iff the following three properties hold [25]:*

(i) The most significant non-zero bit is positive.

(ii) Among any w consecutive digits, at most one is nonzero.

(iii) Each non-zero digit is odd and less than 2^{w-1} in absolute value.

Similar to NAF, wNAF form assures a unique form of an integer, but the hamming weight is even smaller. The density of a wNAF key is $\frac{1}{w+1}$, on average 1 in every $w+1$ symbols is nonzero.

Table 2.1: wNAF Forms for 1622

Binary		1	1	0	0	1	0	1	0	1	1	0
NAF[1]	1	0	$\overline{1}$	0	1	0	$\overline{1}$	0	$\overline{1}$	0	$\overline{1}$	0
3NAF		0	3	0	0	1	0	$\overline{3}$	0	0	3	0
4NAF		0	3	0	0	0	0	5	0	0	$\overline{5}$	0

[1]Here a "-" sign over a digit means negating the digit, eg, $\overline{3}$ denotes -3m.

2.3.2 Generation of wNAF

The algorithm to generate a wNAF form binary is as follows: [16], note the term *mods* is defined as: x *mods* y returns a value j, and $y/2 \leq j \leq y/2$, and $j \bmod y = x \bmod y$. For example, 7 *mods* $2^4 = 7$, while 9 *mods* 2^4 is not equal to 9 but -7, since 7 is greater than $2^4/2$.

Algorithm 4 Generation of wNAF

input width w, and n-bit integer d

output width w, $\{\sigma_n, \sigma_{n-1}, \cdots, \sigma_0\}_w\}$ is wNAF of d

1: $i \leftarrow 0$
2: **while** $d > 0$ **do**
3: **if** d is even **then**
4: $\sigma_i \leftarrow 0$
5: **else**
6: $\sigma_i \leftarrow d mods\ 2^w$
7: $d \leftarrow d - \sigma_i$
8: **end if**
9: $d \leftarrow d/2; i \leftarrow i + 1;$
10: **end while**
11: **RETURN** $\{\sigma_n, \sigma_{n-1}, \cdots, \sigma_0\}_w\}$

The following shows how to compute the scalar multiple kP using a wNAF form of a key.

Algorithm 5 Scalar Multiple with wNAF Key

input Base point P, secret key k

output $Q = kP$

1: **for** $x \leftarrow \pm 1, \pm 3, \cdots, \pm 2^{w-1}$ **do**
2: Compute $K[x] \leftarrow xP$
3: **end for**
4: Set $Q \leftarrow O, i \leftarrow n - 1$
5: **while** $i \geq 0$ **do**
6: $Q \leftarrow 2Q$
7: **if** $k_i \neq 0$ **then**
8: $Q \leftarrow Q + K[k_i]$
9: **end if**
10: **end while**
11: **RETURN** Q

3. SMART CARD SECURITY

A Smart Card, also called chip card or integrated circuit card, is a plastic or paper made pocket-size card embedded with an integrated circuit. A smart card provides a tamper-proof storage of cardholders identity and account information. Smart Cards are used in a variety of applications. Typically they play a role in some type of proof/identification, i.e. in an authentication scheme. In an application, there is a reader that interacts with the card and processes the data. The systems enhanced with smart cards covers although the industries, such as banking, telecommunication, E-commerce, health care, etc.

3.1 Architecture of Smart Card

The shape of smart card is restricted by International Organization for Standardization (ISO), and the card that is the most often referred to is CR80, defining the size, thickness, memory volume, but this is not the only type of card that is deployed today. Special shape of cards are being heavily used, such as the cutoff card: SIM card, Micro-SIM, and Mini-SIM.

The categorization of the smart card may lie in applications it is used for, such as credit cards, satellite TV cards, cell phone cards, etc. In industry, the differences can rely on the operation system. Two primary types of OS for a smart card are: fixed file structure system and dynamic application system, which depends on the requirement of particular applications. Another way to characterize smart cards is from a security point of view, two primary type of smart card are: Symmetric Key Encryption Capable and Asymmetric Key Encryption Capable. The defining difference lies in the encryption capability of the OS and chip.

3.2 Hardware Architecture

Smart cards' functionality are embodied in a single small chip, with components as follows: [33]

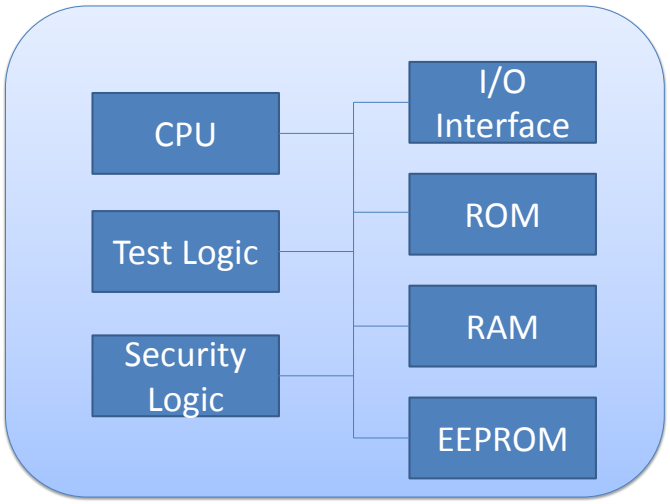


Fig. 3.1.: Smart Card Architecture

CPU(Central Processing Unit) The core of the chip, and all the computations, including some related cryptographic processing, is completed within this part. The frequency for the mainstream of current smart cards is 3.57MHz.

Test Logic A logic component to test the manufacturing errors.

Security Logic A verification function to test the environment of the card which probability affects the security.

I/O Interface A functionality to accept external commands and send responses according to particular communication protocols.

ROM The permanent memory, containing the operating system and self-testing API. The size is 32KB for most smart card.

RAM The re-writable memory for temporary values, such as session keys, internal variables, stack data, etc. The size is typically 1kHz.

EEPROM Non-volatile updatable memory used for application data, which is depending on the application type, eg, keys, PINs, phone number, balances, etc. Also used for operating system and application code.

3.3 Smart Card Reader and Terminals

The smart card readers and terminals are devices used to access the card and obtain data or perform a transaction. There are two types of cards, the defining difference is basing on if the reading process requires a physical connection with the chip. The former one is called contact-less card, and the latter one is contact card. The paper [30] provides an excellent overview on the cards.

Contact-less Smart Card The chip on this type of card is usually hidden, which involves electromagnetic induction to provide power from terminal to chip. When the card comes closer to the reader, a radio frequency that helps the communication. Many of contact-less card are designed for Physical Access Control, Transposition Applications, and payment, such as Electronic Toll Collection (ETC) system being used on Express way. The dominant protocol under the ISO 14443 is MIFARE, follows by EMV standards [30].

Contact Smart Card [33] This type of card requires a physical connection with the card. From the Fig 3.3.

1. **The Vcc and Ground** Provide the power.
2. **Reset** Facilitates a hard restart of all processes.
3. **Vpp** Originally provided a higher EEPROM programming voltage, which is no longer in use.
4. **I/O** This contact is a seriel channel for bi-directional communication between chip and the reader.

Fig. 3.2.: Smart Card Architecture

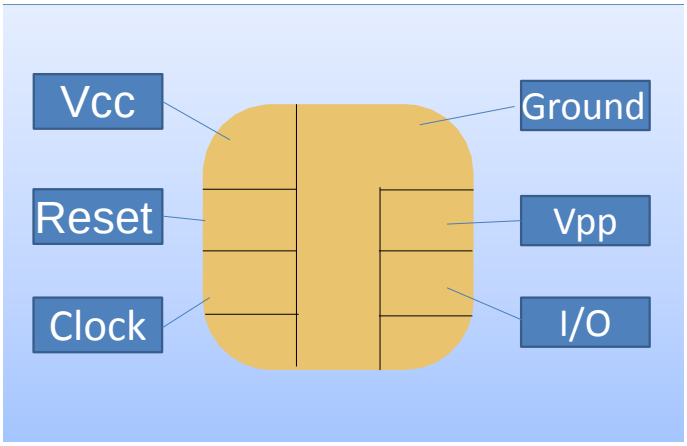


Fig. 3.3.: Contact Card Connections

3.4 Smart Card Security

Smart cards provide enormous benefits of portable and tamper-proof storage of card-holders identity and account information. As people access sensitive information, which occurs in a large number of applications, the use and importance the security of smart card rises.

Security is needed to protect the information to be stolen, lost or altered. In addition, according to formation and process in the implementation, security factors both in time and space will be involved:

Security in Time Smart card is accessed when the data needed to be created, updated, exchanged and stored via networks. The vulnerability within these processes need to be considered and get rid of through those essential steps.

Security in Space In implementation of the smart cards, all of these elements below are suspected to contain possible vulnerabilities:

1. **Hardware** Possible physical parts for smart card could be servers, storage devices, communication channels, etc. All the devices possibly get the data from the system could be attacked.
2. **Software** Operating system, database management software and even security management software itself.
3. **Personnel** Any human who could access to the hardware or software, including professional personnel, administrative personnel, hardware or software related staff.

3.4.1 Data Integrity

Data integrity is necessary to maintain the accuracy and consistency of data over the entire lift-cycle. For smart card, needs this function to verify the data while creating and updating. For example, a smart card could store the parent-to-child relationship records. The data-integrity mechanism is such that no child record can exist without related parent. This assures when a child's information is inputted requiring a parent information. This also prevents an unexpected deletion of a parent's data which is related to the child's data.

3.4.2 Authentication

The process insures the identity of people involved in the transaction. The strength of authentication relies on how many factors are used to confirm the identity. For example a employee ID authentication usually involves the ID number and date of birth, maybe the name as well. The more factors involved, the less vulnerability to the authentication there is. While this will also debase the efficiency and the user experiences. Some speed-up mechanisms are often applied in this process.

3.4.3 Non-Repudiation

Non-repudiation is a service that provides proof of the integrity and origin of data, for which the user cannot later deny. In this setting, the authentication can be asserted to be genuine with high assurance. To achieve this, one needs to provide a proof of data integrity in such a way that it cannot be later denied by the prover. A digital signature is a common tool used.

3.4.4 Data Security Algorithm

Smart card involves two types of encryption algorithm:

Symmetric Key Encryption [30] The sender and receiver shares the same key. The most famous symmetric encryption algorithm is Data Encryption Standard (DES), which was standardized by NIST on the 1970s. To lengthen the use of DES and satisfy the current security level for symmetric key cryptography, triple-DES was developed, DES used with with three encryptions. The number of keys could be three or two [20]. AES (Advanced Encryption Standard) is also a commonly used symmetric encryption [31].

Asymmetric Key Encryption There is another category of asymmetric key encryption.

3.5 Threads to Smart Card

For the card issuer, such as the bank, E-commerce company, etc., there are typically two ways of using smart card, *host based* or *card based*.

In host based scheme, smart card is simply a data carrier. All the protection is done from the host computer, while in the card based smart card is teated more as a microprocessor. Both of them rely critically on security services such as authentication, data secrecy, data integrity, non-repudiation, etc.

The valuable and sensitive information is owned by the card issuer rather than the card possessor. The possessor of the card can attempt to maliciously access these secret information. Once the signing key is known to the adversary they can construct a signature and bypass the necessary procedure, a result in a banking application could imply unlimited access to the banks financial resources.

E-banking and e-commerce businesses are using smart cards to authenticate users and secure online transactions. Hardware devices that are often used as security tokens. Implemented in a number of applications with diverse security requirements, such as banking and identification applications, the smart card may be used to provide a signature, including applications that rely critically on security services such as authentication, data secrecy, data integrity, non-repudiation, etc. This valuable and sensitive information is owned by the card issuer rather than the card processor. The possessor of the card can attempt to maliciously access these secret information. Once the signing key is known to the adversary they can construct a signature and bypass the necessary procedure, a result in a banking application could imply unlimited access to the banks financial resources.

3.6 Smart Card Attacks

Smart cards are small, cheap, portable and easy to get. Moreover, the enormous sensitive usage, turn this tiny chip into a popular target for hackers. Manufacturers and attackers are always challenging each other, while new advances are constructed, but there is no 100% secure smart card. [33] There are three basic attacks: logical, physical and Side-Channel attacks.

3.6.1 Logical Attack

In a logical attack, the attacker could try to exploit bugs in the following areas.

Software Implementation A smart card is a microprocessor. Excluding those commnds it must perform to run, the chip supports thousands of additional commands.This functionality may be abused for undesired data retrievals and modifications.

Hidden Commands Smart card system theoretically supports more than 65000 commands, though only a few of them are required for a particular application. The rest could be abused.

Parameter Positioning and Buffer Overflow By inputing an invalid parameter value, which may be not allowed or exceed the length, will result in surprised result. For example, if the input overflows the buffer size.

Crypto-Protocol, Design and Implementation Cryptographic protocols handless the cryptographic operations on smart card. If the protocol is not carefully designed, this hidden flaws may be affect the normal behavior of chip. For example, some cards have fallback methods to enhance reliability in case of technical problems, while this is right insecure and attacker may benefit from creating fictitious functions.

3.6.2 Physical Attack

In a physical attack, the attacker could try to exploit bugs in the following areas. An overview of physical attacks is provided in [33].

Hardware This can be achieved by high-end lab equipment.

Chemical Solvents, Etching and Staining Material This is to decapsulate and delayer the chip to reveal the building blocks. Then an optical and electrical analysis can be involved to take a close look at the chip.

Optical and Electron Microscope Although the size of chip feature is below 1 micron, it is still visible through a electron, even an optical microscope. A carefully designed chip still can be analyzed to reveal its active sections, running code and even values on the data bus.

Probe Stations A tiny probe is placed on an arbitrary wire to produce a new channel. If the data bus is able to be located via the above two approaches, all the data transfer can be intercepted, eg, programing code, programing data, including keys.

Contactless Attack As it is mentioned before, smart card can communicate with the terminal with any physical connection using the Radio-Frequency Identifier (RFID). The effective range for RFID is around 10cm. Kfir and Wool [15] show a picking virtual pocket using relay attacks on this contactless smartcard system, that is to trick the reader into communicating with a victim smartcard that is far away. The attack system contains two parts, the "ghost" and "leech". The leech is 50cm away from the card, doing the reading. The data will send afar to the ghost. And the ghost can be up to 50 meters away from the card reader with 3 orders of magnitude higher than the normal, working as a middle terminal transmit the data to the reader. This scheme provides an unlimited distance for the attack on the RFID system.

3.6.3 Side Channel Attack

A side channel attack attempts to exploit some physical phenomena to analyze or modify the smart card behavior, such as time, power, etc.

Many attacks on smart cards have been categorized as side-channel attacks. In reality, cryptographic algorithms are always implemented in software or hardware on physical devices with interact with and are influenced by their environments. The physical interactions can be investigated and monitored by adversaries, and may result in information useful in cryptanalysis. This type of information is called sign channel information and the attacks exploiting side-channel information are called side-channel attacks [34].

3.6.4 Passive Attack

Some side channel attacks are passive attacks, which attempt to learn information about the key by listening to some side-channels without interfering with the computation, in a word, it is attempting to break the system solely based upon observing data, e.g, voltage, timing, etc. Kocher [18] introduces a timing attack on implementation of Diffie-Hellman, RSA, DSS (Digital Signature Standard), and other systems. By closely monitoring the running time of signing operations, people can easily find the property of secret key, this kind of attack is not time consuming at all.

Timing Attack Since the operations of execution of programing code varies, timing attack monitors the CPU, memory, simply based on comparing how much time each operation takes. Then analyze what exactly the code is. Statistic is involved in this attack.

Electromagnetic Analysis (EMA) EMA is used widely in smart card attacks. It is based on leaked electromagnetic radiation which can directly provide secret information of chip. To develop this type of tools requires advanced knowledge of hardware and signal processing. Such measurements can also be used in power analysis.

3.6.5 Active Attack

Another type of attack is an active attack, which is, persistent attempt to introduce invalid data into a system, and/or to damage or destroy data already stored in it. In many countries, it is a criminal offense to attempt any such action.

A fault attack is a side-channel attack which is an active attack. The fault attack is caused by inducing a fault into the smart card, while the device is executing the program. The adversary then observes all channels of information, including the output, in an attempt to recover information about the key. These attacks are fundamentally different from passive side-channel attacks. In [6]Boneh, DeMillo and Lipton constructed a fault attack (Bellcore Attack) demonstrating how a malicious user can recover the secret key information from the card. In this paper we discuss fault attacks on an essential cryptographic elliptic curve computation, as well as RSA.

3.6.6 Sign Change Fault Attack

Fault attack is a type of side channel attack in the field of cryptography. The principle is to induce faults-unexpected environmental conditions-into cryptographic implementations, to reveal internal states and/or faulty outputs.

In earlier work, fault attacks on ECC worked on moving the scalar multiplication to a different or even weaker curve. However, this can be easily detected by verifying the parameters and that the resulting point belongs to the original curve. The attack described in Blömer, Otto and Seifert [5] results in a faulty point on the original curve. This attack involves a *sign change* on Q (the accumulator) as described in

Algorithm 2. In their attack, Q becomes $-Q$ at some iteration i. In their work they used Theorem 1, as well as the following notation. Above we are adopting the following notation. If k is the key then for each i, where $0 \leq i \leq n$, define $L_i(k) = \sum_{j=0}^{i} 2^j k_j$ and $H_i(k) = \sum_{j=i}^{n} 2^j k_j$. Thus $L_i(k)$ represents the low $i+1$ bits of key k and $H_i(k)$ represents the high $n-i+1$ bits of the key k. Consequently $k = L_n(k) = H_0(k)$.

Theorem 1. *Let $\mathcal{B}$ be a set of n balls numbered 1 to n. Let m be an integer where $m \ll n$ and C be an integer where $1 \leq C \leq n$. Suppose we randomly select C many balls from $\mathcal{B}$ without replacement. Let A be the event that for every interval I of length m of the interval $[1, n]$ there exists a ball b with $b \in I$ that was selected. If $C \geq \frac{n}{m} \log 2n$ then $Prob(A) \geq \frac{1}{2}$.*

Assuming one is using an algorithm like Algorithm 2, then the faulty output $\tilde{Q}$ can be represented as:

$$\begin{aligned} \tilde{Q} &= (-H_i(k) + L_{i-1}(k))P \\ &= (-H_i(k) + -L_{i-1}(k) + L_{i-1}(k) + L_{i-1}(k))P \\ &= -Q + 2L_{i-1}(k)P. \end{aligned}$$

Then the attacker makes sufficient number C of faulty sign change attacks, here they used the constant $C = \frac{n}{m} \log 2n$. They then execute an off-line analysis, using their table of c many faults, predicting keybits from lowest to highest. By computing C many faults they satisfy that the probability that every interval I of 0 to $n-1$, which has length m, has a faulty $\widetilde{Q}$ with the fault at $i \in I$ exceeds $1/2$. Thus they will be able to compute the secret key by executing this procedure on average twice. Here m is a parameter such that exponential work in m is feasible as an off-line computation.

3.6.7 Counter Fault Attack on ECC

A novel fault attack is described in King and Wang [17]. During the execution of Algorithm 2, a fault is injected into the calculation and implicitly lands on line 2. That is, $i \leftarrow i-1$ was skipped at one iteration. This is equivalent to alter the original k to the faulty k denoted as $\hat{k}$ as below:

$$\hat{k} = k_{n-1} \cdots k_{i+1} k_i k_i k_{i-1} \cdots k_1 k_0.$$

Instead of computing $Q = kP$, the output will be:

$$\tilde{Q} = \hat{k}P = 2Q + 2^{ik_i}P - L_{i-1}(k)\,P. \tag{3.1}$$

The Equation 3.1 can also be expressed as:

$$\tilde{Q} = \hat{k}P = H_i(k)\,P + 2^i k_i P + Q. \tag{3.2}$$

This indicates that we can also achieve the same goal by guessing the highest bits to lowbits.

In [17] they introduced the concept of a *valid pattern*. Because faults produce key expressions of the form $k_{n-1}k_{n-2}\ldots k_{i+1}k_ik_ik_{i-1}\ldots k_1k_0$, and because a faulty output is of the form $\widetilde{Q} = 2Q + 2^i k_i P - L_{i-1}(k)P$, they characterized that a *valid pattern* ("guess") $x_j x_{j-1} x_{j-2}, \ldots$ should be tested with the term of $(-2^j x_j + 2^{j-1}x_{j-1} + 2^{j-2}x_{j-2} + \cdots)P$. They observed that $0xxx\ldots$ is the same as $0000xxx\ldots$, in the sense of the testing value $(-2^j x_j + 2^{j-1}x_{j-1} + \cdots)P$. If the first string is deemed to be successful as a "good guess" then the second string will also be successful. One needs to consider patterns that have a leading zero, this is needed due to the fact that fault could have been generated at some i for which $k_i = 0$. Thus they [17] noted that a guess should either have the leading bit which is nonzero (so we test with $(-2^j + 2^{j-1}x_{j-1} + \cdots)P$ or the leading bit is zero but the next bit is nonzero. Also, there is a problem concerning guesses with a leading bit which is a one. For example, given the guess 11101, they would test with $(-2^4 \cdot 1 + 2^3 \cdot 1 + 2^2 \cdot 1 + 2 \cdot 0 + 1)P = -3P$. At the same time, for valid pattern 101 they will test with $(-2^2 + 1)P = -3P$. Thus

for different valid patterns they would use the same test value, there is no way we can distinguish if the keybits were 101 or 11101. So they found that for valid patterns with a leading one, they could not determine multiple keybits, they did note they could determine one keybit, that the low-bit of the pattern is the correct keybit.

Definition 2. *[17] A* **valid pattern** $x = x_\nu, \ldots, x_1$ *is a binary sequence of length* ν *where* $1 \leq \nu \leq m$ *such that either* (i) *we have* $x_\nu = 1$ *or* (ii) *we have* $\nu \geq 2$, $x_\nu = 0$ *and* $x_{\nu-1} = 1$.

Notice that "all zeros" is not a possible valid pattern, in the King et. al. [17] construction. If an adversary is convinced that a fault has occurred on an interval of length m, and all valid patterns fail then the adversary knows that a sequence of zeros occurs in the keystream. The adversary cannot determine the number of zeros, but they do know at least one zero occurred.

Algorithm to Recover the Bits

So the adversary induces C many counter faults and stores them in the database. Initially the adversary knows no key bits so they set $r = -1$, $r + 1$ is the number of known key bits. Because there are C faults, then [17] every interval of length m has a fault. They start looking at all valid patterns, once they form a guess they compute $G_x = (-2^j x_j + 2^{j-1} x_{j-1} + \cdots + x_0)P$ if there exists a faulty output $\widetilde{Q}$ such that $G_x + \widetilde{Q} = 2Q$ then they found key bits $x_j, .., x_0$. If all key bits fail then $k_0 = 0$. They increment r appropriately. At the next iteration they compute G_x as:

$$G_x = L + (-2^{r+\nu} x_{r+\nu} + \sum_{j=r+1}^{r+\nu-1} x_j 2^j)P.$$

Here L is the $r+1$ know lower keybits scalar multiple $L = L_{r+1}(k)P$. Lastly, once the number of "known keybits" exceeds $n-m$ then we cannot claim that there is a fault in $\mathcal{B}$ for which the bit position i occurs in $\{n-m+1, \ldots n-1\}$. However we can determine the remaining keybits by exhaustive search, since there are at most $m-1$ many unknown bits. This would require at most 2^{m-1} steps (and m was selected so that 2^m steps is reasonable).

Algorithm 6 The Counter Fault Attack

1: *Let* $B \leftarrow (n/m)\log(2n)$
2: *Create* B faulty outputs of Algorithm 2 by inducing a fault on counter i, where i occurs uniformly random on $\{0, \ldots, n-1\}$, so that it prevents a decrementation
3: *Let* $\mathcal{B} = \{\widetilde{Q}_{f_j} : f_j \text{ is the } j^{th} \text{ counter fault of Algorithm 2 with inputs } k \text{ and } P, j = 1, \ldots, B\}$
4: *Set* $r \leftarrow -1$ {here $r+1$ represents the number of least significant bits of k known by the attacker, initially $r + 1 = 0$}
5: **while** $r \leq n - m$ **do**
6: *Let* $L \leftarrow \mathcal{O} + \sum_{j=0}^{r} k_j 2^j P$ {here keybits $\{k_j : j < r\}$ of the secret key k are known by the adversary}
7: **for** all *valid patterns* $x = x_{r+\nu}, \ldots, x_{r+1}$ (see Definition 2) **do**
8: *Let* $\mathcal{G}_x \leftarrow L + (-2^{r+\nu} x_{r+\nu} + \sum_{j=r+1}^{r+\nu-1} x_j 2^j)P$
9: **for** $\widetilde{Q} \in \mathcal{B}$ **do**
10: {The following is the verification step and tests if valid pattern x is correct}
11: **if** $(\mathcal{G}_x + \widetilde{Q}) = 2Q$ **then**
12: **if** $r + 1 \leq 0$ **then**
13: $W \leftarrow$ Algorithm 7$(x, q, P, \mathcal{G}_x, \tilde{Q}, Q)$
14: **if** $W \neq NIL$ **then**
15: **output** W {Comment: In this case W equals the key k}
16: **end if**
17: **else**
18: **if** $x_{r+\nu} = 0$ **then**
19: *Conclude* $k_{r+1} = x_{r+1}, \ldots, k_{r+\nu} = x_{r+\nu}$
20: *Set* $r \leftarrow r + \nu$ and continue **go to** line 5.
21: **else**
22: *Set* $k_{r+1} = x_{r+1}$ and let $r \leftarrow r + 1$ and continue **go to** line 5.
23: **end if**
24: **end if**

Algorithm 6 Counter Fault Attack cont'd

```
25:         end if
26:       end for
27:     end for
28:     if no valid pattern satisfies the verification step then
29:       Set k_{r+1} = 0 and let r ← r + 1
30:     end if
31: end while
32: if r < n then
33:     At this point the adversary knows key bits k_r ..., k_1 k_0 where r > n − m. By
        exhaustive search determine k_{n−1}, ..., k_{r+1} so that Q = kP.
34:     if exhaustive search fails to produce Q = kP then
35:       return "failure"
36:     end if
37: else
38:     if Q ≠ kP then
39:       return "failure"
40:     end if
41: end if
42: output k
```

3.6.8 Counter Fault Attack on RSA

We implemented the algorithm and applied it to RSA. During this implementation we discover an error. The problem is King and Wang [17] never took into account the possibility of multiple consecutive occurrences of all valid patterns failing, which causes a 0 as the known keybits, if this happened for example three consecutiveness times then we will generate 000, but consecutive zeros is problematic. We solved this by constructing the following state machine for the counter fault attack.

Algorithm 7 Algorithm to Recover k When $r+1=0$, and $x_{r+\nu}=1$ or $x_{r+\nu}=0$

Input : $a, q, P, \mathcal{G}_x, \tilde{Q}, Q$

1: Let $b = q - a$ {Comment: Recall that to enter this algorithm we have $\mathcal{G}_x + \tilde{Q} = 2Q$ and $r+1=0$. Here $aP = \mathcal{G}_x$ where $a = -2^{r+\nu}x_{r+\nu} + \sum_{j=r+1}^{r+\nu-1} 2^j$. There are two cases to consider: $x_{r+\nu}=1$ and $x_{r+\nu}=0$ }
2: **if** $x_{r+\nu}=1$ **then**
3: **if** $bP = -Q$ **then**
4: return $-b \mod q$ {Comment: In this case $k_i = 0$ and $i = n-1$, so $b = -k$ }
5: **end if**
6: **else**
7: **if** $(b - 2 \cdot 2^{n-1})P = -Q$ **then**
8: return $-(b - 2 \cdot 2^{n-1}) \mod q$ {Comment: this is the case $i = n-1$ and $k_{n-1} = 1$, so $-(b - 2 \cdot 2^{n-1}) \mod q = k$}
9: **end if**
10: **end if**
11: return *NIL*

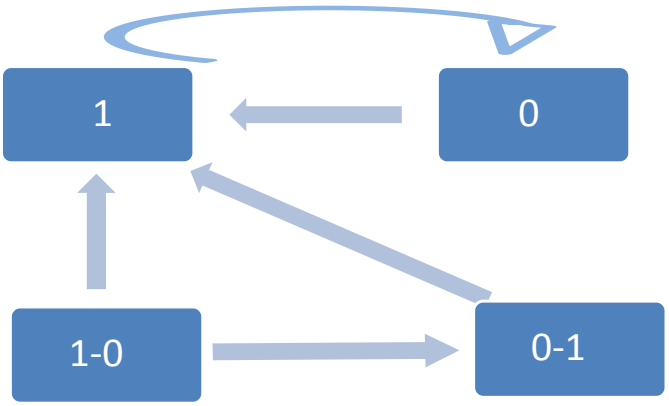


Fig. 3.4.: State Machine for Counter Fault Attack

The state machine in counter fault attack with regular keys possesses four states: “0” state, “1” state, “01” state, and “10” state.

The algorithm to apply the counter fault attack to RSA is as follows: The adversary creates a fault to the counter (i) during the computation of $Sig = M^k \mod N$. The adversary does NOT NEED to know the iteration count where the fault takes place. $Sig = Sig \cdot Sig \cdot M^{2^i k_i - L_{i-1}(k)}$

We implemented the algorithm to attack RSA using the Indiana University supercomputer Quarry. Our results are provided in Fig 3.5.

Algorithm 8 Counter Fault Attack

Make $\frac{n}{m}log(2n)$ faulty signature faulty signatures faultySig
$s = -1$ ($s + 1$ represents the number of known keybits)
while $s < n - m$ **do**
  compute $L = M^{\sum_{j=0}^{s} 2^j k_j}$
  for $r = 0 uptom$ **do**
    for all valid patterns $(x_{s+1}, x_{s+2}, x_{s+r})$ **do**
      $G_x = L * M^{\sum_{j=r+1}^{r+v-1} 2^j x_j - 2^{r+v} x_{r+v}}$
      if $G_x \cdot faultySig = Sig^2$ **then**
        $k_{r+1} = x_{r+1}, k_{r+2} = x_{r+2}, \cdots, k_{r+v} = x_{r+v}$
        $s = s + r$
      end if
    end for
    if no valid patterns **then**
      set $k_{r+1} = 1, s = s + 1$
    end if
    $r = r + 1$
  end for
end while
brute force complete remaining m bits of k

RSA modulus length (bits)	Secure until Year	Window Size(m)	Number of Faults (C)	Running Time(* Indiana Univ. super computer Quarry)
1184	2013	10	1127	60 sec
2048	2023	20	1229	5 minutes
4096	2051	20	2663	35 minutes
6172	2069	20	4194	3 hour 26 minutes

Fig. 3.5.: Simulation Results for Counter Fault Attack on RSA Using a Supercomputer

4. NEW ATTACKS

4.1 Analysis: the Plausibility of Sign Change Fault Attack

This attack involves a changing the sign of an elliptic curve point. Blömer et. al. discussed the use of a key expressed in NAF form. The use of a NAF representation does not increase the chances of the sign change attack as the attack is on the memory of the point and not on a program instruction. In an implementation of ECC one would not use a sign bit to represent the ECC point. Further in the various fields that are used to build an ECC application, the negative of a point may not need a sign bit. Recall in $GF(2^m)$, if $P = (x, y)$ then $-P = (x, y + x)$, here addition is an xor, no sign bits needed. In $\mathbb{Z}_p$, if $P = (x, y)$ then $-P = (x, -y)$, thus a negative could be used. But it would never be used in an efficient implementation, as $-y$ is computed modulo p. Thus $-y = p - y \bmod p$. Hence a negative would never be used and certainly not used in $GF(2^m)$.

Consequently the impact to force Q to become $-Q$ would be a significant series of field operations and difficult to generate via a fault. One could attempt to infer that this could occur based on the use of a NAF representation, but causing a fault on an instruction based on k being 1 or -1 would not be sufficient to cause the accumulator to go from Q to $-Q$.

Lastly one could view the Sign Change attack as a key stream attack perhaps as the key traveled on the bus from memory to CPU. But the modification of the key along this transmission would be highly selective. For example a sequence in NAF form (it is not necessary to be in NAF form):

$$1, 0, -1, 0, 1, 0, 0, 0, -1, 0, 0, 1, 0, 1, 0, 0, 1$$

would be modified to:

$$-1, 0, 1, 0, -1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1$$

So that only 1 and -1 are modified for all $j > i$, 0's remain unchanged.

We now describe two more plausible fault attacks, one on the key stream and the other on a program instruction.

4.2 Bit Flip Attack BFA

As described earlier, the Sign Change attack can be argued as an attack on the key stream where key bits (belonging to $\{-1, 0, 1\}$ are flipped in sign. If b is a bit, let $flip\,(b)$ denote the flipped bit. Thus $b \oplus flip\,(b) = 1$, where $\oplus$ denotes an xor. We now discuss the `BIT FLIP` attack. In this case we assume that the key stream $k_n k_{n-1} \ldots k_2 k_1 k_0$ is attacked such that there exists an i, for all $j \geq i$, $\hat{k}_j = flip\,(k_j)$ and for all $j < i$, $\hat{k}_j = k_j$.

Example 1. *Suppose k is given by $k = k_{17} \ldots k_0 = 101001101001101001$ and there is an attack at $i = 13$, then the key stream, denoted by $\hat{k}$ is:*

$$\hat{k} = \hat{k}_{17} \ldots \hat{k}_{13} \hat{k}_{12} \ldots \hat{k}_0 = \hat{k}_{17} \ldots \hat{k}_{13} k_{12} \ldots k_0 = 010111101001101001.$$

Thus $Q = kP = \sum_{i=0}^{n} k_i 2^i P$ and we represent $\hat{k}P$ by $\widetilde{Q}$ then:

$$\begin{aligned}
\widetilde{Q} &= \sum_{j=0}^{n} \hat{k}_j 2^j P \\
&= \sum_{j=0}^{i-1} \hat{k}_j 2^j P + \sum_{j=i}^{n} \hat{k}_j 2^j P \\
&= \sum_{j=0}^{i-1} k_j 2^j P + \sum_{j=i}^{n} flip(k_j) 2^j P \\
&= \sum_{j=0}^{i-1} k_j 2^j P + \sum_{j=i}^{n} flip(k_j) 2^j P + \sum_{j=i}^{n} k_j 2^j P - \sum_{j=i}^{n} k_j 2^j P \\
&= L_{i-1}(k)P + (2^{n+1} - 2^i)P - H_i(k)P
\end{aligned}$$

$$
\begin{aligned}
&= L_{i-1}(k)P + (2^{n+1} - 2^i)P - H_i(k)P \\
&= L_{i-1}(k)P + L_{i-1}(k)P - L_{i-1}(k)P + (2^{n+1} - 2^i)P - H_i(k)P \\
&= (2L_{i-1}(k) + 2^{n+1} - 2^i)P - Q.
\end{aligned}
$$

Therefore if $\widetilde{Q}$ is a faulty scalar multiple, produced by the bit flip attack (BFA) then there exists i such that:

$$Q = \left(2L_{i-1}\left(k\right) + 2^{n+1} - 2^i\right) P - \widetilde{Q}. \tag{4.1}$$

An important question to be considered: for a given $\widetilde{Q}$ will the "i" be unique? Suppose for a given faulty output $\widetilde{Q}$, there exist i and i', $i \neq i'$, such that:

$$\widetilde{Q} = (2L_{i-1}(k) + 2^{n+1} - 2^i)P - Q = (2L_{i'-1}(k) + 2^{n+1} - 2^{i'})P - Q. \tag{4.2}$$

This would imply that:

$$\left(2L_{i-1}\left(k\right) - 2^i\right) P = \left(2L_{i'-1}\left(k\right) - 2^{i'}\right) P, \tag{4.3}$$

as we are working over prime subgroup this implies the scalar must be equivalent mod q, i.e. $2L_{i-1}(k) - 2^i = 2L_{i'-1}(k) - 2^{i'}$. This implies that $2L_{i-1}(k) + 2^{i'} = 2L_{i'-1}(k) + 2^i$.

Let's assume without loss of generality that $i > i'$.

Table 4.1: Comparion Fault at i and at i' Where $i > i'$

	i			i'		
$2^{i'}$	0	0	$\cdots$	1	$\cdots$	0
$2L_{i-1}(k)$	*	*	$\cdots$	*	$\cdots$	0
2^i	1	0	$\cdots$	0	$\cdots$	0
$2L_{i'-1}(k)$	0	0	$\cdots$	x	$\cdots$	0

By Table 4.1, we see there are two cases to consider when $2L_{i-1}(k) + 2^{i\prime} = 2L_{i\prime-1}(k) + 2^i$. One, if $x_{i\prime} = 1$ (of $2L_{i\prime-1}(k)$) then $*_{i\prime} = 0$ (of $2L_{i-1}(k)$) and $*_j = 0$ for $i\prime < j < i$ and $*_i = 1$. Second, if $x_{i\prime} = 0$ (of $2L_{i\prime-1}(k)$) then $*_j = 1$ (of $2L_{i-1}(k)$) for $i\prime \leq j < i$ and $*_i = 0$. For example, we see we will have trouble distinguishing $1000**$ with $0001xx$ and $0111**$ with $0000xx$, respectively. However if both $1000**$ and $0001xx$ are correct then $*_1 = x_1$ and $*_2 = x_2$.

Example 2. *Note that if we tested a sequence like 011111 (here $i = 5$) with a faulty scalar multiple $\widetilde{Q}$ and the result was a hit, i.e.$\widetilde{Q} = (2*(011111)P + 2^{n+1}P - 2^5P - Q$, then we know that there may exist a sequence of the form $000***$but which? That is, 000000, 000001, 000011, 000111, and 001111 are all possible illustrations of a i' where $i' = 0, 1, 2, 3, 4$ respectively. For example if i=5 and L_{i-1} represented by 011111 and $i' = 3$ and $L_{i'-1}$ is represented by 000111, then 0111110 +001000 = 0001110+100000.*

Thus by the above example we see we must test multiple cases. We define binary sequences $01111\cdots1*\cdots*$, $10\cdots0*\cdots*$, $0\cdots01*\cdots*$ and $0\cdots0*\cdots*$ as *D-Patterns*. For each D-Pattern, we define its *complementary pattern* by the above observation. Thus 1000** with 0001** are complementary patterns and 0111** with 0000** are complementary patterns. A given D-Pattern may have multiple complementary patterns. For example, the D-Pattern 011111 has complimentary patterns 000000, 000001, 000011, 000111, and 001111. Thus if a D-Pattern like 011111 tests as correct, according to Equation (4.1), then one should test if any one of 000000, 000001, 000011, 000111, and 001111 is correct. For example, if 000111 is also correct then we know the two lowest bits 11 are keybits, we call these the *low bits of the complementary pattern*. Also we will have a problem if the faulty output occurs

on the first bit position of the interval, in the sense we are testing for $L_{i-1}(k)$ and using 2^i, thus if $i = 0$ then $L_{i-1}(k)$ is void/empty. We work around the problem by searching over an interval of length $m + 2$. Our algorithm, see Algorithm 14, is very similar to the algorithm by Blömer et. al. [5]. We will generate $c = \frac{n}{m} \log 2n$ many BIT FLIP faults. By computing c many faults they satisfy that the probability that every interval I of 0 to n, which has length m, has a faulty $\widetilde{Q}$ with the fault at $i \in I$ exceeds $1/2$, again this uses a result by Boneh et. al. in [6].

4.3 Bit Flip Attack on RSA

The previous section referred to the *BFA* on ECC. What if it is applied to the RSA system, with scalar multiple substituted by exponentiation.

$$C = M^{\left(2L_{i-1}(k)+2^{n+1}-2^i\right)} \cdot \widetilde{C}^{-1}. \tag{4.4}$$

Then:

$$\widetilde{C} = M^{\left(2L_{i-1}(k)+2^{n+1}-2^i\right)} \cdot C^{-1}. \tag{4.5}$$

If there is $i = i'$ giving the same output $\widetilde{C}$, then:

$$M^{\left(2L_{i-1}(k)+2^{n+1}-2^i\right)} \cdot C^{-1} = M^{\left(2L_{i'-1}(k)+2^{n+1}-2^{i'}\right)} \cdot C^{-1}. \tag{4.6}$$

Since C^{-1} on both side is the same which equivalent to:

$$M^{\left(2L_{i-1}(k)+2^{n+1}-2^i\right)} = M^{\left(2L_{i'-1}(k)+2^{n+1}-2^{i'}\right)}. \tag{4.7}$$

then $\left(2L_{i-1}\left(k\right) + 2^{n+1} - 2^i\right) = \left(2L_{i'-1}\left(k\right) + 2^{n+1} - 2^{i'}\right)$ mod p. As it is a subgroup of a prime, we can conclude $\left(2L_{i-1}\left(k\right) + 2^{n+1} - 2^i\right) = \left(2L_{i'-1}\left(k\right) + 2^{n+1} - 2^{i'}\right)$, which is $\left(2L_{i-1}\left(k\right) - 2^i\right) = \left(2L_{i'-1}\left(k\right) - 2^{i'}\right)$. But if the modulus N is composite, then $\left(2L_{i-1}\left(k\right) + 2^{n+1} - 2^i\right) = \left(2L_{i'-1}\left(k\right) + 2^{n+1} - 2^{i'}\right)$ mod $\varphi(N)$.
This is equavalent to:

$$2\left(L_{i-1}\left(k\right) - L_{i'-1}\left(k\right)\right) = \left(2^i - 2^{i'}\right) \text{ mod } \varphi(N).$$

The modulus is composite, where the solution is no longer unique. Let's name the left hand side of the equation as LHS, and the right hand side as RHS. As $i \geq$ i', LHS = RHS + kφ, where k is non-negative integer. $L_{i-1}(k) - L_{i'-1}(k)$ has i bits, so its double, LHS, is $i+1$ bits long. While the LHS is $i-1$ bit long. If $i < n-1$, then $i+1 < n$. LHS couldn't be greater than ϕ, k can only be 0. If $i = n-1$, then $i+1 = n$. LHS could be $n+1$ bits long, k can be either 0 or 1. But this only happens when our guessing moves on to the highest bit. In the algorithm, the last several bits could be recovered by a brute force testing.

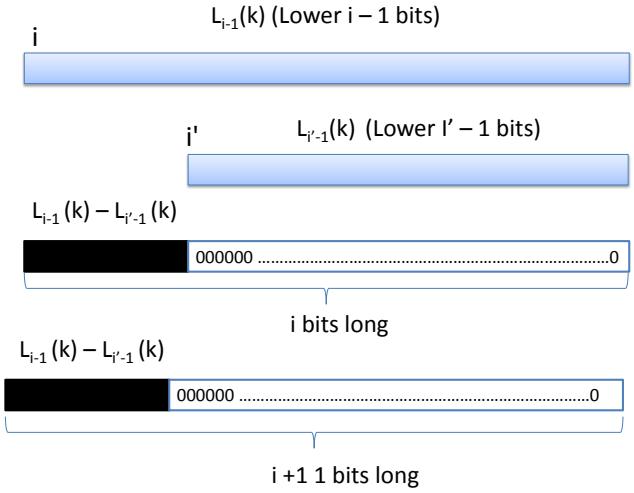


Fig. 4.1.: $2\left(L_{i-1}(k) - L_{i'-1}(k)\right)$

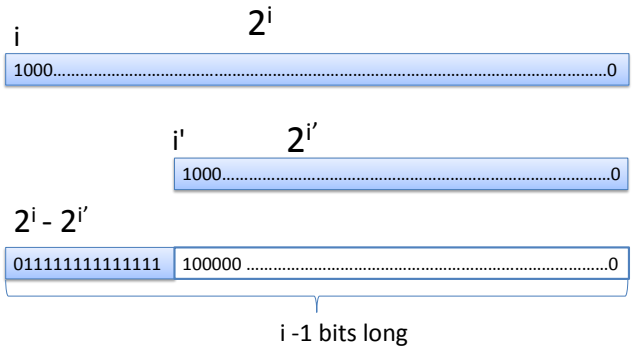


Fig. 4.2.: $\left(2^i - 2^{i'}\right) \bmod \varphi(N)$

Outline of Bit Flip Attack

Algorithm 9 Outline of Bit Flip Attack

1: Initialize $Key = 0$. Let m denote a suitable window size, we attempt to recover at most m bits at a time. Thus 2^m work must be feasible.
2: Create $C = \frac{n}{m}log(2n)$ many faulty scalar multiples using Bit Flip attack, location of each fault is random. Store each faulty output in a database.
3: **while** entire *key* is not known **do**
4: **for** each "suitable pattern" where the length of pattern , satisfies $\leq m+1$ **do**
5: compare it to each faulty output in the database.
6: **if** Equation 4.1 is satisfied **then**
7: record the known *key* bits (will be $\leq$)
8: increment the known key bit counter by the appropriate value.
9: **end if**
10: **end for**
11: **end while**

4.3.1 Running Time of Algorithm

The number of faults to be injected in the system is $\frac{n}{m}log(2n)$, as we mentioned in Chapter 3. The running time of algorithm depends on length of key n and width of "window" m. After each successful guess, we update our *key*, and increment the search position, and we are guaranteed to increase our search position by one for every iteration. As there are at most n iterations, we could conclude the running time to be:

$$T = O(C \times \frac{n}{m} \times 2^m).$$

$$T = O(\text{key length} \times \text{number of faults} \times \text{ number of valid patterns} \times \text{pattern width}). \tag{4.8}$$

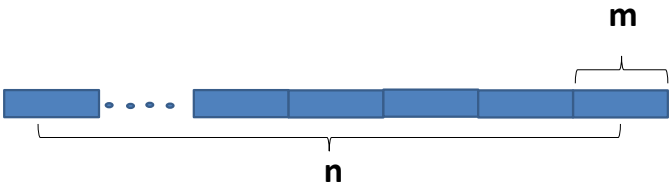


Fig. 4.3.: Running Time

Worst Case Analysis

The worst case is, when moving on with each chunk, has to go over all the possible patterns and find an agreement. Then the number of patterns for each window size is $2^1 + 2^2 + \cdots + 2^{m-1} + 2^m$, which is $2(2^m - 1)$ in summation. Hence:

$$T = O(C \times \frac{n}{m} \times 2^m) = O(C \times \frac{n}{m} \times 2(2^m - 1)) = 2C \times n\frac{2^m - 1}{m}.$$

When m increases, we see T increases. If we do differentiate T with respect to m, then:

$$\frac{dT}{dm} = 2 \times C \times n \times (2^{m-1} - \frac{2^m - 1}{m^2}).$$

Best Case

When moving the window with each successful discovery of keybits we see:

$$T = O(C \times \frac{n}{m} \times 1) = O(\frac{C \times n}{m}).$$

Average Case

The average case is that it is needed to go over half of the all valid patterns for each m bits, which is $\frac{2(2^m-1)}{2} = 2^m - 1$. Then:

$$T = O(C \times \frac{n}{m} \times (2^m - 1)).$$

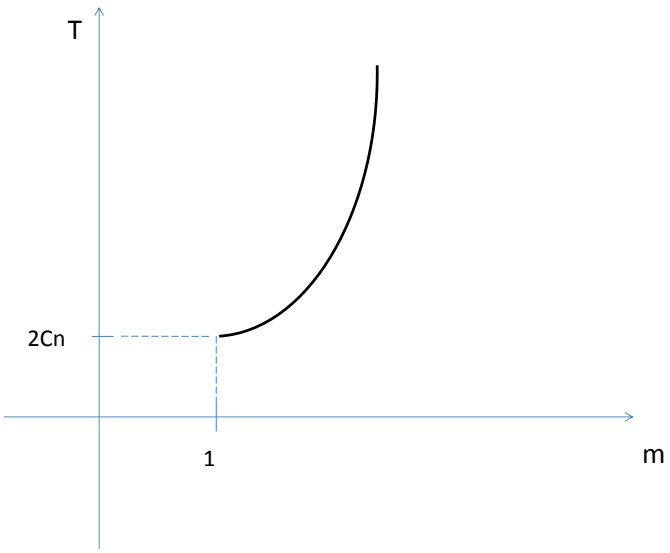


Fig. 4.4.: Running Time Plot of Worst Case

Algorithm 10 Bit Flip Attack

Let $c \leftarrow \frac{n}{m}\log(2n)$

Create c BIT FLIP faulty outputs of Algorithm 2 by inducing a BIT FLIP fault on counter i, where i occurs uniformly random on $\{0, \ldots, n\}$,

Let $\mathcal{B} = \{\widetilde{Q}_{f_j} : f_j$ is the j^{th} BIT FLIP fault of Algorithm 2 with inputs k and P, $j = 1, \ldots, c\}$

Set $r \leftarrow -1$ {here $r+1$ represents the number of least significant bits of k known by the attacker, initially $r + 1 = 0$}

while $r \leq n - m$ **do**

  Let $L \leftarrow \mathcal{O} + \sum_{j=0}^{r} k_j 2^j P$ {here keybits $\{k_j : j < r\}$ of the secret key k are known by the adversary}

  for all binary sequences $x = x_{r+\nu}, \ldots, x_{r+1}$ of length $\nu < m + 2$ **do**

    Let $\mathcal{G}_x \leftarrow L + (\sum_{j=r+1}^{r+\nu} x_j 2^j)P$

    for all $\widetilde{Q} \in \mathcal{B}$ **do**

      if $(2\mathcal{G}_x - 2^{r+\nu+1})P = Q - 2^{n+1}P + -\widetilde{Q}$ **then**

        if $x = x_{r+\nu}, \ldots, x_{r+1}$ is a D-Pattern **then**

          for all complementary patterns ω of x **do**

            if complementarity pattern ω when concatenated to L tested according to Equation (4.1) is proved to be correct **then**

              $r = r +$ number of *low bits of the complementary pattern*

              insert *low bits of the complementary pattern* into key stream k and update L

              break out of for loop

            end if

          end for

        end if

      else

        $r = r + \nu$

        $k_{r+\nu} = x_{r+\nu}, \ldots, k_{r+1} = x_{r+1}$

      end if

    end for

Algorithm 10 Bit Flip Attack cont'd

25: **end for**

26: **end while**

27: **if** $r < n$ **then**

28: At this point the adversary knows key bits $k_r \ldots, k_1 k_0$ where $r > n - m$. *By exhaustive search determine* $k_n, \ldots, k_{r+1}$ so that $Q = kP$.

29: **if** exhaustive search fails to produce $Q = kP$ **then**

30: **r**eturn "failure"

31: **end if**

32: **else**

33: **if** $Q \neq kP$ **then**

34: **r**eturn "failure"

35: **end if**

36: **end if**

37: **output** k

4.4 Doubling Attack

In our second attack we assume that the adversary can generate a fault on step 4 of Algorithm 2 so that the doubling of Q is skipped for some iteration i. The execution bypasses the instruction $Q = 2Q$ and proceeds to executing the key bit test if $k_i == 1$ to determine if an add is necessary. Thus all key bits are executed but they are not added properly, and the output is faulty given by $\widetilde{Q}$. The analysis is given by:

$$\begin{aligned}\widetilde{Q} &= (2^{n-1}k_n + \cdots + 2^{i-1}k_i + 2^{i-1}k_{i-1} + \cdots + 2k_1 + k_0)P \\ &= H_i(k)\frac{1}{2}P + Li - 1(k)P \\ &= H_i(k)\frac{1}{2}P + L_{i-1}(k)\frac{1}{2}P + L_{i-1}(k)\frac{1}{2}P \\ &= \frac{1}{2}Q + L_{i-1}(k)\frac{1}{2}P.\end{aligned}$$

Here $\frac{1}{2}Q$ denotes the point V such that $2V = Q$.

To illustrate this attack, consider the following example. Suppose we are computing kP where k is given by 110101101001. The scalar multiple $\hat{k}P$ that is actually calculated, is described by the following Table 4.2.

Table 4.2: Illustration of Doubling Attack

k	1	1	0	1	0	1	1	0	1	0	0	1
after fault									1	0	0	1
consequence of fault		1	1	0	1	0	1	1	0			

As in the 8th iteration, Q is prevented from being doubled, which is equivalent to shift all the higher 8 bits to the right by one,where the 8th bit is overlapped by the 7th bit(count from the MSB). Hence:

$$\hat{k} = 11010110 + 1001 = 11010111001.$$

Denote the points $1/2P$ by P_{half} and $1/2Q$ by Q_{half}. Then:

$$\widetilde{Q} = Q_{\mathsf{half}} + L_{i-1}(k)P_{\mathsf{half}}. \tag{4.9}$$

Since the keyspace is $\mathbb{Z}_q$, for prime q, the integer $\frac{q+1}{2}$ is such that $2 \cdot \frac{q+1}{2} = 1 \bmod q$. Thus $2^{-1} = \frac{q+1}{2}$. Hence $Q_{\mathsf{half}} = \frac{q+1}{2}Q$ and $P_{\mathsf{half}} = \frac{q+1}{2}P$. An alternate way to compute the half of an ECC point. On the right hand side of Equation 4.9, there are several elements that need to be determined: P_{half}, Q_{half}. The half of a point can be computed by the logarithm of Knudsen [35]. Let $P = (x, y)$ then P can be represented as $P = (x, \lambda)$ where $\lambda = x + \frac{y}{x}$, rather than affine coordinates $P = (x, y)$. Let $W = (u, \lambda_W)$ represent the half of point P, i.e. $W = \frac{1}{2}P$. We denote P by $P = (x, \lambda_P)$. Then λ_W can be determined by solving:

$$\lambda_W^2 + \lambda_W = a + x. \tag{4.10}$$

Once one solved for λ_W, the x-coordinate can be determined by first computing:

$$tmp = x(\lambda_W + \lambda_P + x + 1). \tag{4.11}$$

Now observe that $TRACE(a + x)$ must equal 0, which is true if and only if P is the double of some point, an observation that is used in both [35]. Recall that there are two solutions to Equation 4.10, how do we know if we have computed the correct solution, i.e. is λ_W correct? (The other solution is $\lambda_W + 1$.) For this setting, if λ_W is correct then the x-coordinate of W is $\sqrt{tmp}$. However if λ_W is incorrect, i.e. the correct coordinate is $\lambda_W + 1$ then x-coordinate of W is $\sqrt{tmp + x}$ (here x is the x-coordinate of P). How do we determine the correct λ? This is achieved by noting that the point is such that the computed x coordinate must satisfy a trace condition. Further $TRACE(tmp) = TRACE(\sqrt{tmp})$. So if $TRACE(tmp + a) = 0$ we see that λ_W is correct, otherwise we should use $\lambda_W + 1$. This is summarized in Algorithm 11.

The details of this attack are very analogous to those of BIT FLIP attack. The test condition to search for is $L_{i-1}(k)P_{\mathsf{half}} = \widetilde{Q} - Q_{\mathsf{half}}$.

Algorithm 11 Halving a point

Input $P = (x, \lambda)$

Output $W = HALF(P)$

1: Solve for λ_W in the equation $\lambda_W^2 + \lambda_W = a + x$.
2: $tmp = x(\lambda_W + \lambda + x + 1)$
3: **if** $TRACE(tmp + a) \neq 0$ **then**
4: $\quad tmp = tmp + x$
5: $\quad \lambda_W = \lambda_W + 1$
6: $\quad u = \sqrt{tmp}$
7: **end if**
8: $u = \sqrt{tmp}$
9: return $W = (u, \lambda_W)$

4.4.1 For a Given $\widetilde{Q}$ Will the "i" be Unique?

Again, suppose for a given faulty output $\widetilde{Q}$, there exists i and i', $i \neq i'$, such that:

$$\widetilde{Q} = Q_{\mathsf{half}} + L_{i-1}(k)P_{\mathsf{half}} = Q_{\mathsf{half}} + L_{i'-1}(k)P_{\mathsf{half}}. \tag{4.12}$$

Since Q_{half} is a constant, then:

$$L_{i-1}(k)P_{\mathsf{half}} = L_{i'-1}(k)P_{\mathsf{half}}.$$

Here P_{half} is a nonzero constant too, which implies:

$$L_{i-1}(k) = L_{i'-1}(k),$$

as we are working over prime subgroup this implies the scalar must be equivalent. So the value of $L_{i-1}(k)$ and $L_{i'-1}(k)$ are equal. There are two possibilities here: first, $i = i'$; Second, $i > i'$, then $x_i, x_{i-1} \cdots x_{j+1}$ are all equal to 0. We can still conclude there is no collision for recovered bit stream.

Doubling Attack on RSA

As we mentioned before, RSA involves exponentiation instead of scalar multiple. Hence the half of P and Q becomes $\sqrt{P}$ and $\sqrt{Q}$. In this system:

$$P_{\mathsf{half}} = \sqrt{P},$$
$$Q_{\mathsf{half}} = \sqrt{Q} = \sqrt{P^d},$$

where d is an odd integer, so we cannot use $P^{\frac{1}{2}d}$ as the result. Need to find the square root named "quadratic residue" for P and Q. However, the ability to compute square roots modulo a composite is equivalent to factoring, which is infeasible to the adversary. Consequently, due to the infeasibility of computing a square root the Double Attack cannot be executed for the RSA cryptosystem.

Algorithm 12 Outline of Doubling Attack

1: Compute $\frac{1}{2}P$ and $\frac{1}{2}Q$.
2: Initialize $Key = 0$. Let m denote a suitable window size, we attempt to recover at most m bits at a time. Thus 2^m work must be feasible.
3: Create $C = \frac{n}{m}log(2n)$ many faulty scalar multiples using Doubling attack, location of each fault is random. Store each faulty output in a database.
4: **while** entire key is not known **do**
5: **for** each "suitable patterns where the length of pattern satisfies $\leq$ m + 1 **do**
6: compare it to each faulty output in the database.
7: **if** Equation 4.9 is satisfied **then**
8: record the known key bits
9: increment the known key bit counter by the appropriate value.
10: **end if**
11: **end for**
12: **end while**

Algorithm 13 Algorithm of Doubling Attack

Input : Access to Algorithm 1, n is the length of secret key $k > 0$ in non-adjacent form, Q is the correct result which equals to kP, m is the interval of bits recovering and is also the acceptable amount of offline work. We precompute half of P and Q to store them in $halfP$ and $halfQ$.

Input : k

1: Set $C = (n/m)\log(2n)$
2: Compute Q
3: Compute $HP = \frac{q+1}{2}P$
4: Compute $HQ = \frac{q+1}{2}Q$
5: Induce c faulty outputs of Alg. 1 applying doubling attack of Q for a random i.
6: Store the collection all $\widetilde{Q}$ into $\mathcal{C}$
7: Set $s = -1$ (s is the starting location to guess)
8: **while** $s < n$ **do**
9: Compute $L = \sum_{j=0}^{s} 2^j k_j HP$
10: **for** (i=0 ; r¡=m; r++) **do**
11: **for** all NAF form of $x_{s+1}, x_{s+2}x_{s+r}$ **do**
12: Compute $Tx = L + \sum_{j=s+1}^{s+r} 2^j x_j HP$
13: **if** $Tx - \widetilde{Q} = HQ$ **then**
14: $k_{s+1} = x_{s+1}, k_{s+2} = x_{s+2}, \ldots, k_{s+r} = x_{s+r}$
15: set $s = s + r//s + r = I$; go to line 5
16: **end if**
17: **end for**
18: **end for**
19: **if** no Tx satisfy $s = s + 1$ **then**
20: Go to line 5
21: **end if**
22: **end while**
23: Verify $Q = KP$ if = output k else output failure

4.4.2 Running Time of Algorithm

The algorithm for doubling attack is similar to bit flip attack, except there is an extra computation on half points. But since it is a one-time operation for the attack. So the computational complexity is constant. Thus running time of doubling attack will also be:

$$T = O(C \times \frac{n}{m} \times 2^m)$$

with:

$$C = (\log 1/(2n))/(\log(1m/n))$$

number of faults. The worst case, best case and average case are the same as bit flip attack.

5. COUNTER FAULT ATTACK ON WNAF KEY

Observe that the *Counter Fault attack* has difficulty handling a sequence of consecutive zeros in the key stream. Thus one may attempt to compute the kP using a key with many consecutive zeros. For example, expressing the key k in NAF or wNAF form and computing kP. When we are trying to attack the cryptosystem, we are blind to the value of the key but we are also blind to the form of the key. It could be in regular binary form, Non-adjacent Form NAF, or even wide-ary Non Adjacent Form (wNAF). It is well known if a key is in NAF form then the density of the key is approximating 1/3rd. A key in wNAF form is an even more sparse, here the density of the wNAF key is $1/(w+1)$.

In wNAF, the key is represented using a symbol set of 0 together with non-zero odd values which are in absolute value less than 2^{w-1}. By definition 2NAF is NAF and uses the symbols $-1, 0, 1$. For 3NAF, there are 5 symbols to consider $-3, -1, 0, 1, 3$. In general, a key in $wNAF$ form has nonzero symbols, on average, one in every $w+1$ positions. Thus if a counter fault attack is applied in this setting we see the previous attack may not be successful. Here we address this problem. Taking integer 619 as an example, the different representations of it are as follows:

Table 5.1: wNAF Forms for 619

binary		1	0	0	1	1	0	1	0	1	1
NAF[1]		1	0	1	0	0	$\overline{1}$	0	$\overline{1}$	0	$\overline{1}$
3NAF	1	0	0	$\overline{3}$	0	0	0	$\overline{3}$	0	0	3
4NAF		1	0	0	0	0	7	0	0	0	$\overline{5}$

[1]Here a "-" sign over a digit means negating the digit, eg, $\overline{1}$ denotes -1.

5.1 Attacking A Cryptographic Computation When the Key is in wNAF Form

Note the attack on the cryptographic calculation can be conducted on an ECC scalar multiple, a discrete log calculation (exponentiation), or an RSA signature calculation. We illustrate this work using a discrete log calculation.

In the case of elliptic curve and discrete log calculations, the keyspace is a prime field. In the case of RSA the keyspace is $\mathbb{Z}_{\varphi(N)}$ which is not a field. We now consider the Counter fault attack, based on King et. al. algorithm, to recreate it, the use of additional nonzero symbols that occur in wNAF representation is the main emphasis. In the Counter Fault attack [17], we generate a fault to block line 7 in Algorithm2, that is the instruction $i \leftarrow i - 1$ is not executed for one iteration. As noted, the original key has length n so key:

$$k = k_{n-1}k_{n-2}\cdots k_{i+1}k_ik_{i-1}\cdots k_1k_0$$

becomes, due to this fault:

$$\hat{k} = k_{n-1}\cdots k_{i+1}k_ik_ik_{i-1}\cdots k_1k_0.$$

Since k_i is repeated, the higher bits from k_{i+1} to k_{n-1} are shifted left by one, and the lower i bits remain the same. Then the faulty output of a $C = M^e \bmod N$ calculation will be:

$$\begin{aligned}
\tilde{C} &= M^{\hat{k}} \\
&= M^{(k_{n-1}k_{n-2}\cdots k_{i+1}k_i ki-1\cdots k_1k_0)} \\
&= M^{(k_{n-1}k_{n-2}\cdots k_{i+1}k_i)} \cdot P^{(k_i ki-1\cdots k_1k_0)} \\
&= M^{2H_i(k)} \cdot M_i^L(k) \\
&= (M^{H_i(k)})^2 \cdot M^{(L_{i-1}(k)+2^i k_i)} \\
&= (M^{H_i(k)})^2 \cdot M^{(L_{i-1}(k)} \cdot M^{2^i k_i} \\
&= C \cdot M^{H_i(k)} \cdot M^{2^i k_i} \\
&= C^2 \cdot M^{2^i k_i - L_{i-1}(k)}.
\end{aligned}$$

Hence:

$$\tilde{C} \cdot M^{L_i(k)-2^i k_i} = C^2. \tag{5.1}$$

For example, given a key $k = 661$, then k can be written in 3NAF as `003000300-3`. Now if a fault occurs at the 2^{nd} bit, then the faulty key:

$\hat{k}$ will be $00300030003\overline{3}$.

Table 5.2: Example of Fault on Zero

k		0	0	-3	0	0	0	3	0	0	-3
$\hat{k}$	0	0	-3	0	0	0	3	0	0	0	-3

We now consider possible attack scenarios. From Equation 5.1, the faulty output $\widetilde{C}$ is:

$$\tilde{C} = C^2 \cdot M^{2^i k_i - L_{i-1}(k)}.$$

Since in this example $k_i = 0$, then:

$$\tilde{C} = C^2 \cdot P^{-L_{i-1}(k)}.$$

Let $G_x = M^{L_{i-1}(k)}$. Then $G_x \cdot \tilde{C} = C^2$.

Now suppose the fault occurs at some nonzero symbols, such as the 4^{th} bit, then the faulty key $\hat{k}$ will be:

Table 5.3: Example of Fault on Nonzero

k		0	0	-3	0	0	0	3	0	0	-3
$\hat{k}$	0	0	-3	0	0	0	3	3	0	0	-3

$$\tilde{C} = C^2 \cdot M^{2^i k_i - L_{i-1}(k)}.$$

Since $k_i \neq 0$, then the approach of the counter fault attack, i.e. check if $G_x \cdot \tilde{C} = C^2$ is no longer valid. Instead, there must be a check $G_x \cdot M^{-2^i k_i} \cdot \tilde{C} = C^2$. That is, we need to consider a multiplication on the right hand side of the equation. The counter fault attack algorithm requires an update concerning the detection.

Thus if one knows the "w" that was used in the wNAF exponentiation and believes the fault occurred at a $k_i \neq 0$ then one has to check:

$$\tilde{C} = C^2 \cdot M^{2^i x - L_{i-1}(k)}, \tag{5.2}$$

where one has to check for all:

$$\tilde{C} \in \text{database}.$$

and for all:

$$x \in \{\pm 1, \pm 3, \ldots, \pm 2^{w-1} - 1\}.$$

Of course this means they know where the fault occurred, at i, and they know all keybits below the fault position. Both extremely unlikely, but this observation helps us derive the attack.

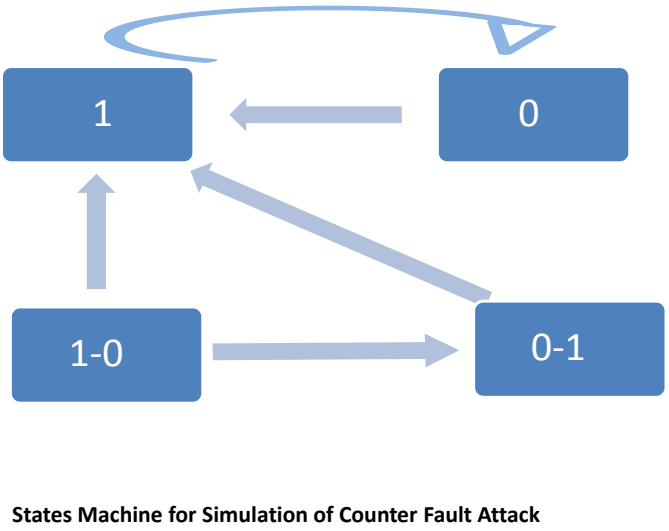


Fig. 5.1.: State Machine for Simulation of Counter Fault Attack on no NAF Key

5.2 Algorithm for Counter Fault Attack on RSA with wNAF Key

From the analysis above, we see that the main scheme for determining a fault for a wNAF key calculation is the same as the original attack on the binary key calculation except when fault lands on non zeros. In such situations the value G_x needs to be multiplied by a factor $M^{-2^i k_i}$. This is one essential point in the revised algorithm. Secondly, the symbols that are used are no longer only 0 and 1. While implementing the line 10 of Algorithm 5.1, we need to consider the symbol with multiple possibilities using an extra for loop iterating w from $-2^{w-1}+1$ to $2^{w-1}-1$. The state machine in counter fault attack with regular keys possesses four states: 0 state, 1 state, 01 state, and 10 state.

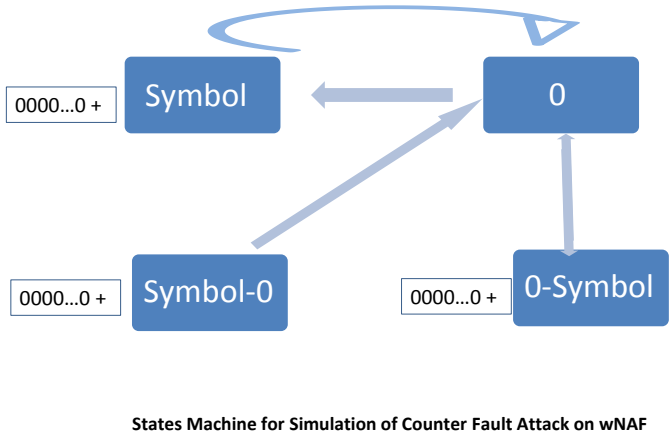


Fig. 5.2.: State Machine for Simulation of Counter Fault Attack on wNAF Key

Now for wNAF representation of keys, the symbols are more varied. For example in 3-NAF will have "0" state, "1" state, "$\overline{1}$" state, "3" state, "$\overline{3}$" state, "01" state, "0$\overline{1}$" state, "10" state, "$\overline{1}$0" state, "03" state, "30" state, "0$\overline{3}$" state, "$\overline{3}$0" state, which gives us 13 states in total. There are even more for 4-NAF, etc. So we generalize all nonzeros in to one object and call it "symbol". Hence for all wNAF, there are four states: "0" state, "symbol" state, "0symbol" state, and "symbol0" state. The state machine and algorithm can be recreated as above:

Algorithm 14 Counter fault attack on RSA with wNAF Key

input width w, $\hat{Q}$,key length n, window size m, P,Q **output** k

1: *Let* $B \leftarrow \frac{n}{m} log(2n)$

2: *Create* B faulty outputs of Algorithm 2 by inducing a fault on counter i, where i occurs uniformly random on $\{0, \ldots, n-1\}$, so that it prevents a decrementation

3: *Let* $\mathcal{B} = \{\widetilde{Q}_{f_j} : f_j$ is the j^{th} counter fault of Algorithm 2 with inputs k and P, $j = 1, \ldots, B\}$

4: *Set* $r \leftarrow -1$ {here $r+1$ represents the number of least significant bits of k known by the attacker, initially $r + 1 = 0$}

5: **while** $r \leq n - m$ **do**

6: *Let* $L \leftarrow P^{\sum_{j=0}^{r} k_j 2^j}$ {here keybits $\{k_j : j < r\}$ of the secret key k are known by the adversary}

7: **for** all *valid patterns* $x = x_{r+\nu}, \ldots, x_{r+1}$ (see Definition 2) **do**

8: *Let* $\mathcal{G}_x \leftarrow L \cdot P^{(-2^{r+\nu} x_{r+\nu} + \sum_{j=r+1}^{r+\nu-1} x_j 2^j)}$

9: **for** all possible symbol of the leading bit, from $-2^{w-1} + 1$ to $2^{w-1} - 1$ **do**

10: **for** all $\widetilde{Q} \in \mathcal{B}$ **do**

11: {The following is the verification step and tests if valid pattern x is correct }

12: $\mathcal{G}_x \leftarrow \mathcal{G}_x \cdot P^{-symbol \ll r+\nu}$

13: **if** $(\mathcal{G}_x \cdot \widetilde{Q}) = Q^2$ **then**

14: **if** $r + 1 \leq 0$ **then**

15: $W \leftarrow$ Algorithm 7$(x, q, P, \mathcal{G}_x, \tilde{Q}, Q)$

16: **if** $W \neq NIL$ **then**

17: **output** W {Comment: In this case W equals the key k}

18: **end if**

19: **else**

20: **if** $x_{r+\nu} = 0$ **then**

21: *Conclude* $k_{r+1} = x_{r+1}, \ldots, k_{r+\nu} = x_{r+\nu}$

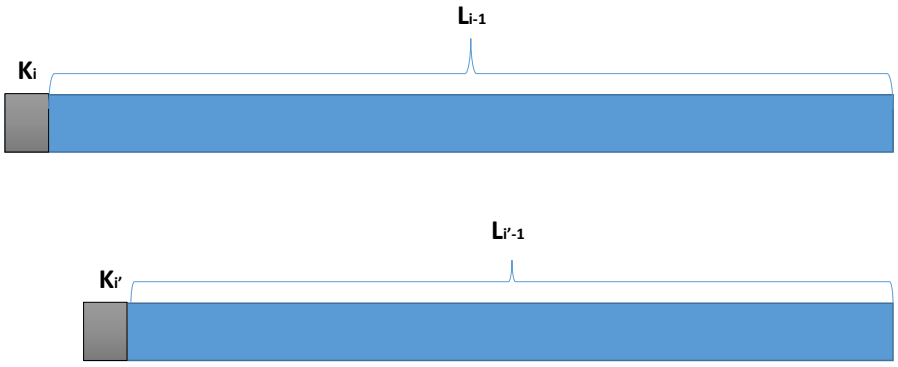
22: *Set* $r \leftarrow r + \nu$ and **go to** line 5.

23: **else**

24: *Set* $k_{r+1} = x_{r+1}$ and let $r \leftarrow r + 1$ and **go to** line 5.

Algorithm 14 Counter fault attack on RSA with wNAF Key cont'd

25: **end if**
26: **end if**
27: **end if**
28: **end for**
29: **end for**
30: **end for**
31: **if** no valid pattern satisfies the verification step **then**
32: *Set* $k_{r+1} = 0$ and let $r \leftarrow r + 1$
33: **end if**
34: **end while**
35: **if** $r < n$ **then**
36: At this point the adversary knows key bits $k_r \ldots, k_1 k_0$ where $r > n - m$. *By exhaustive search determine* $k_{n-1}, \ldots, k_{r+1}$ so that $Q = P^k$.
37: **if** exhaustive search fails to produce $Q = P^k$ **then**
38: **return** "failure"
39: **end if**
40: **else**
41: **if** $Q \neq P^k$ **then**
42: **return** "failure"
43: **end if**
44: **end if**
45: **output** k



5.3 Uniqueness of the Recovered Bits

An important question, similar to the questions asked in the bit flip attack, for a given $\widetilde{C}$ will the "i" be unique? Suppose for a given faulty output $\widetilde{C}$, there exists i and i', $i \neq i'$, such that:

$$\widetilde{C} = C^2 \cdot M^{2^i k_i - L_{i-1}(k)} = C^2 \cdot M^{2^{i'} k'_i - L_{i'-1}(k)}. \tag{5.3}$$

This would imply that $M^{2^i k_i - L_{i-1}(k)} = M^{2^{i'} k_{i'} - L_{i'-1}(k)}$. As we are working over the ring $\mathbb{Z}_N$ this implies the exponent must be equivalent $\bmod \varphi(N)$, i.e. $2^i k_i - L_{i-1}(k) = 2^{i'} k_{i'} - L_{i'-1}(k) \bmod \varphi(N)$. All equations are expressed $\bmod \varphi(N)$.

From above we have:

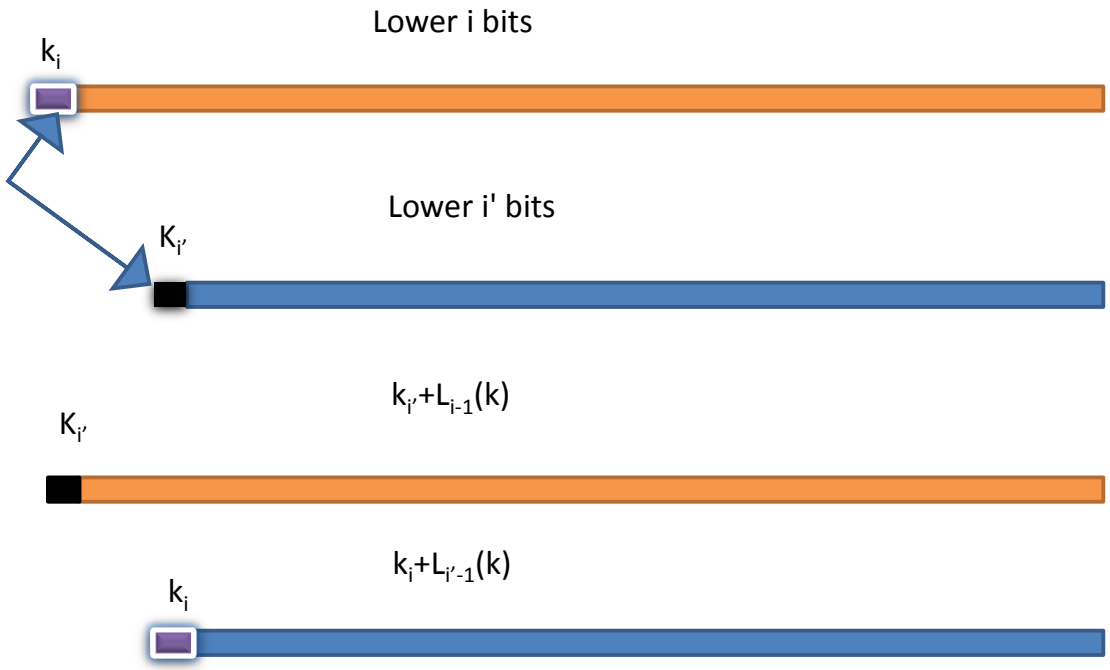
$$2^i k_i - 2^{i'} k_{i'} = L_{i-1}(k) - L_{i'-1}(k). \tag{5.4}$$

Let's assume without loss of generality $i > i'$. We can consider this in four possible cases.

1. If $k_i = k_{i'} = 0$, then from Equation 5.4, we can conclude $L_{i-1}(k) = L_{i'-1}(k)$ and that the two guesses are equivalent, though their length differs;

2. If $k_i = 0, k_{i'} \neq 0$, then $L_{i'-1}(k) - 2^{i'} k_{i'} = L_{i-1}(k)$, which means the lower $i-1$ bits and lower $j-1$ bits have the equal value. Then $i = j$.

3. If $k_i \neq 0, k_{i'} = 0$, can prove i = j in the same way as case 2;



4. If $k_i \neq 0, k_{i'} \neq 0$, since $2^i k_i - 2^{i'} k_{i'} = L_{i-1}(k) - L_{i'-1}(k)$, then $2^i k_i + L_{i'-1}(k) = 2^{i'} k_{i'} + L_{i-1}(k)$.

Assume we have two key streams, the first one is $k_i k_{i-1} \cdots k_1 k_0$, another one is $k_{i'} k_{i'-1} \cdots k_1 k_0$. The left hand side forms a new key stream, $k_i k_{i-1} \cdots k_1 k_0$, which we represent by a. The right hand side forms another key stream, $k_i k_{i'-1} \cdots k_1 k_0$, which we represent by b. From the equation we can see, $a = b$, but both the leading bits $k_{i'}, k_i$ are nonzeros. so a and b must have the same length. Since a and b are just the lower i bits and i' bits with leading digit swapped, so the length of lower i bits and i bits are identical. Hence $i = i'$. In conclusion, i and i' are equivalent.

6. CONCLUSION

In this thesis, we designed two novel fault attack: Bit Flip Attack and Doubling Attack. Also, by implementing King and Wang's Counter Fault Attack for RSA cryptosystem we have collected some significant simulations results. Further, we developed an improvement to the Counter Fault Attack to eliminate an unexpected faulty bit stream. Lastly we have extended the Counter fault Attack so that it can attack keys in NAF and wNAF form.

LIST OF REFERENCES

LIST OF REFERENCES

[1] Accredited Standards Committee X9, American National Standard X9.62-2005, Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA), November 16, 2005.

[2] R. Anderson. "Survey of Smartcard Attack Technologies. Last Date Accessed: April 25, 2014, http://ftp.sunet.se/pub/security/docs/crypt/Ross_Anderson/.temp/report.pdf.

[3] S. Arno, and F. Wheeler. "Signed digit representations of minimal Hamming weight." Computers, IEEE Transactions on 42, no. 8 (1993): pp.1007-1010.

[4] I. Blake, S. Gadiel, and S. Nigel. Elliptic Curves in Cryptography. Vol. 265. Cambridge University Press, 1999.

[5] J. Blmer, M. Otto, and J. Seifert. "Sign change fault attacks on elliptic curve cryptosystems." In Fault Diagnosis and Tolerance in Cryptography, pp. 36-52. Springer Berlin Heidelberg, 2006.

[6] D. Boneh, D. Richard, and L. Richard. "On the importance of checking cryptographic protocols for faults." In Advances in CryptologyEUROCRYPT97, pp. 37-51. Springer Berlin Heidelberg, 1997.

[7] A. Chilver.(2002, April 10). Twenty Years of Smartcards and Smartcard Attacks. Last Date Accessed: April. 14, 2014, http://www.scmagazine.com/twenty-years-of-smartcards-and-smartcard-attacks/article/30679/.

[8] W. Diffie, and M. Hellman. "New directions in cryptography." Information Theory, IEEE Transactions on 22, no. 6 (1976): pp. 644-654.

[9] W. Diffie, and M. Hellman. "Privacy and authentication: An introduction to cryptography." Proceedings of the IEEE 67, no. 3 (1979): pp. 397-427.

[10] T. ElGamal. "A public key cryptosystem and a signature scheme based on discrete logarithms." In Advances in Cryptology, pp. 10-18. Springer Berlin Heidelberg, 1985.

[11] D. Hankerson, S. Vanstone, and A. Menezes."Guide to Elliptic Curve Cryptography, Springer-Verlag", New York 2004.

[12] D. Hardy, F. Richman, and C. Walker. Applied algebra: codes, ciphers and discrete algorithms. CRC Press, 2011.

[13] M. Joye, and S. Yen. "Optimal left-to-right binary signed-digit recoding," IEEE Transactions on Computers, vol. 49, no. 7, pp. 740–748, 2000.

[14] D. G. (2013, October 21). Cryptographic Key Length Recommendation. Last Date Accessed: April 14, 2014, http://www.keylength.com/en/4/.

[15] Z. Kfir, and A. Wool. "Picking virtual pockets using relay attacks on contactless smartcard." Security and Privacy for Emerging Areas in Communications Networks, 2005. SecureComm 2005. First International Conference on IEEE, 2005.

[16] B. King. "wNAF*, an efficient left-to-right signed digit recoding algorithm." Applied Cryptography and Network Security. Springer Berlin Heidelberg, 2008.

[17] B. King, and W. Wang. "Counter Fault Attack: a Hardware Fault Attack. Forthcoming, N.d.

[18] P. Kocher. "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems." Advances in CryptologyCRYPTO96. Springer Berlin Heidelberg, 1996.

[19] A. McCullagh, and W. Caelli. "Non-repudiation in the digital environment." *First Monday*, 5(8)(2000).

[20] A. Menezes, P. Oorschot, and S. Vanstone. Handbook of Applied Cryptography. CRC press, 1996.

[21] F. Mollet. "Card Fraud Nets Esc6 billion," in *Cards International*, (Sept 22, 1995), p. 3.

[22] National Institute of Standards and Technology (NIST). (2014, April 14). Digital Signature Standard, FIPS PUB 186-2. Last Date Accessed: April 14, 2014, http://csrc.nist.gov/publications/fips/fips186-2.htm.

[23] National Institute of Standards and Technology (NIST). (1999, July). Recommended elliptic curves for federal use. Last Date Accessed: April 14, 2014, http://www.nist.gov.

[24] Victor Shoup. (2013, February 15). A Tour of NTL: Introduction. Last Date Accessed: April 14, 2014, http://www.shoup.net/ntl/doc/tour-intro.html.

[25] K. Okeya, Samoa, C. Spahn, and T. Taskagi. "Signed binary representations revisited." Advances in CryptologyCRYPTO 2004. Springer Berlin Heidelberg, 2004.

[26] G. Reitwiesner. "Binary Arithmetic." Advances in Computers 1 (1960): pp. 231-308.

[27] R. Rivest, A. Shamir, and L. Adleman. "A method for obtaining digital signatures and public-key cryptosystems." Communications of the ACM 21, no. 2 (1978): pp. 120-126.

[28] National Institute of Standards and Technology (NIST). (2014, April 14). SHA-2 Standard, Secure Hash Standard, FIPS PUB 180-2. Last Date Accessed: April 15, 2014, www.itl.nist.gov/fipspuhs/fip180-2.htm.

[29] National Institute of Standards and Technology (NIST). (2014, April 14). SHA-1 Standard, Secure Hash Standard, FIPS PUB 180-1. Last Date Accessed: April 15, 2014, www.itl.nist.gov/fipspubs/fip180-1.htm.

[30] CardLogix Corporation. (2010, n.d.). Smart Card Overview. Last Date Accessed: April 14, 2014, http://www.smartcardbasics.com/smart-card-security_2.html.

[31] L. Washington, and W. Trappe. Introduction to cryptography: with coding theory. Prentice Hall PTR, 2002.

[32] Wikipedia. (2014, April 21). Field. Last Date Accessed: April 21, 2014, http://en.wikipedia.org/wiki/Field_(mathematics).

[33] M. Witteman. "Advances in smartcard security." Information Security Bulletin 7, no. 2002 (2002): pp. 11-22.

[34] Y. Zhou, and D. Feng. "Side-Channel Attacks: Ten Years After Its Publication and the Impacts on Cryptographic Module Security Testing." IACR Cryptology ePrint Archive 2005 (2005): pp. 388.

[35] E. Knudsen. "Elliptic scalar multiplication using point halving." In Advances in Cryptology-ASIACRYPT99, pp. 135-149. Springer Berlin Heidelberg, 1999.